Security Amplification of Threshold Signatures in the Standard Model

Karen Azari[1*], Cecilia Boschini[2], Kristina Hostáková[2], Michael Reichle[2]

[1] University of Vienna, Austria
karen.azari@univie.ac.at

[2] ETH Zurich, Switzerland
{cecilia.boschini, kristina.hostakova, michael.reichle}@inf.ethz.ch

Abstract. The current standardization calls for threshold signatures have highlighted the need for appropriate security notions providing security guarantees strong enough for broad application. To address this, Bellare et al. [Crypto'22] put forward a hierarchy of unforgeability notions for threshold signatures. Recently, Navot and Tessaro [Asiacrypt'24] introduced a new game-based definition of *strong* (one-more) unforgeability for threshold signatures, which however does not achieve Bellare's strongest level of security.

Navot and Tessaro analyzed several existing schemes w.r.t. their strong unforgeability security notion, but all positive results rely on idealized models. This is in contrast to the weaker security notion of (standard) unforgeability, for which standard-model constructions exist. This leaves open a fundamental question: is getting strong unforgeability *fundamentally harder* than standard unforgeability for threshold signatures?

In this paper we bridge this gap, by showing a generic construction lifting any unforgeable threshold signature scheme to strong unforgeability. The building blocks of our construction can be instantiated in the standard model under standard assumptions. The achieved notion of strong unforgeability extends the definition of Navot and Tessaro to achieve the strongest level of security according to the hierarchy of Bellare et al. (following a recent classification of security notions for (blind) threshold signatures by Lehmann, Nazarian, and Özbay [Eurocrypt'25]).

The starting point for our transformation is an existing construction for single-user signatures from chameleon hash functions by Steinfeld, Pieprzyk and Wang [RSA'07]. We first simplify their construction by relying on a stronger security notion for chameleon hash functions. The bulk of our technical contribution is then to translate this framework into the threshold setting. Towards this goal, we introduce a game-based definition for threshold chameleon hash functions (TCHF) and provide a construction of TCHF that is secure under DLOG in the standard model. We believe that our new notion of TCHF might also be of independent interest.

* Part of the work was done while the author was at ETH Zurich, Switzerland.

1 Introduction

Threshold signatures [Des88, DF90] allow a subgroup of at least T signers to generate signatures on behalf of $N \geq T$ parties, and guarantee unforgeability as long as at most $T-1$ parties are corrupt. Threshold signatures have a wide range of applications, particularly in the blockchain environment (*e.g.*, in digital wallets), and many schemes have been developed in recent years [KG20, CKM21, RRJ+22, BLT+24, DKM+24]. Their ability to distribute trust among N parties has prompted both IETF [CKGW23] and NIST [BP23] standardization efforts.

For (non-distributed) digital signatures the desired security notion is agreed upon: it should be hard to compute a *non-trivial* forgery σ on a message m given access to a signing oracle. Here, non-trivial means that the message is fresh, *i.e.*, has never been queried to the signing oracle (EUF-CMA). For some applications it is also important that the signature σ itself cannot be tampered with: the signature σ on m is considered non-trivial if the *pair* (σ, m) is fresh, *i.e.*, m might have been queried to the signing oracle but σ was never output by the signing oracle on input m (*strong* EUF-CMA).

In the threshold setting, the adversary can corrupt up to $T-1$ signers and interact with the remaining honest signers via signing oracles (one per honest signer). Again, it should be hard to compute a non-trivial signature, however, the classification of non-triviality is more nuanced and "strength" of the security notion is two-dimensional. As in the non-distributed case, we can distinguish between unforgeability and strong unforgeability. However, in the threshold setting, we have a new dimension of "strength" of the security notion coming from the fact that it is not obvious how to formalize that a message (resp. message-signature pair) is fresh.

Security hierarchy. Many works define a signature σ on message m to be trivial if m was queried to *one* of the signing oracles (*e.g.*, [GJKR96, Bol03, GJKR07, CKM23, BLT+24]). As pointed out by Bellare et al. [BCK+22], this is a rather weak notion. For example, if the adversary A corrupted only a single signer, then intuitively we expect that A must interact with at least $T-1$ honest signers to obtain a valid signature. However, the aforementioned notion already classifies the signature trivial if m was queried for a single honest signer. Shoup [Sho00] gives a stronger security notion that demands that at least $T-C$ honest signers were queried on message m, where C is the number of corrupted signers.

Bellare et al. [BCK+22] give a full hierarchy (increasing in strength) for non-interactive threshold signatures of security notions that carefully classifies which forgeries are considered non-trivial. The weakest notion declares a forgery as trivial if its associated message m has been queried to any signing oracle (as above), that is, if at least one partial signature on m has been generated. The strongest notion in their hierarchy intuitively declares a forgery as trivial only if the adversary completed an *entire* signing session for the message m, that is, every honest party involved in the signing process completed it. In the non-interactive setting (with preprocessing) the concept of completed sessions is formalized through *leader requests:* a leader request specifies the message m, some preprocessing tokens and the set of co-signers $\mathcal{S}$. A forgery is trivial only if

its message m appeared in a leader request that was signed by all honest signers within the leader request's signer set $\mathcal{S}$. Above, preprocessing refers to signing protocols that can be split into a message-independent part and a message-dependant part. Importantly, the former subprotocol can be precomputed offline before the to-be-signed message is determined. This saves valuable resources in the online phase.

Strong unforgeability. To capture non-malleability, Bellare et al. [BCK+22] also define strong unforgeability for non-interactive schemes (in the random oracle model). The recent work by Navot and Tessaro [NT24] provides a more general game-based definition for multi-round schemes based on one-more unforgeability [3]. Here, the adversary is tasked to output ℓ signatures $(\sigma_i)_{i\in[\ell]}$ on some message m. (Note that the game cannot observe the aggregated signatures, therefore demanding that (σ, m) is fresh is not well-defined.) The forgeries are considered non-trivial only if at most $\ell - 1$ signing sessions for message m proceeded to the final round. Note that the counter increases every time a honest party completes a signing session for m, even though it could still be incomplete for some honest users (in contrast to the strongest notion in Bellare et al. [BCK+22]'s hierarchy).

Random oracle model. Bellare et al. [BCK+22] show that FROST1 satisfies strong unforgeability at the second strongest security level in their hierarchy. The same work then also provides a transformation of the scheme to the strongest security notion. The same transformation can also be applied to FROST1-H which was presented and proven strongly secure at the second security level by Tessaro and Zhu [TZ23], again in the random oracle model.

Standard model. For (non-distributed) signatures, it is well-known that strong EUF-CMA security can be achieved in the standard model under non-interactive assumptions (*e.g.*, [BSW06]). For threshold signatures, there are also several constructions in the standard model, however to the best of our knowledge, all such constructions [LJY16, MMS+24] only satisfy weaker unforgeability notions (*i.e.*, non-strong unforgeability and the weakest / second weakest security notion in [BCK+22]'s hierarchy).

On a technical level, this discrepancy seems to stem from the techniques employed to prove stronger unforgeability notions for threshold signatures. In the case of multi-signatures, [NT24] guess a specific random oracle query associated to the forgeries or employ an interactive assumption to facilitate simulation of the signing oracles to achieve strong unforgeability. In this work, we investigate whether such strong assumptions are required to build threshold signatures that satisfy strong security guarantees in the standard model.

[3] One could argue that this scenario is easier to deal with in the UC framework [Can01]. However, as already noted in [NT24] many of the most promising candidates for standardization are not UC-secure, thus the need for game-based definitions.

1.1 Our Contributions

We show that every threshold signature that satisfies the weakest (non-strong) unforgeability notion in Bellare et al. [BCK+22]'s hierarchy can be lifted into a scheme that satisfies the strongest security notion (both with respect to non-malleability and non-triviality of the forgery) *in the standard model*. Notably, we also capture multi-round schemes and strengthen the strong unforgeability notion by [NT24] with respect to its non-triviality classification following the techniques by [BGHJ24, LNÖ25].

Our transformation is entirely in the standard model and relies on a novel building block *Threshold Chameleon Hash Functions* (TCHF) which we introduce in this work. It is a distributed variant of chameleon hash functions (CHF) [KR00], however, the security definitions require much care as we will explain in section 2. We believe that our strengthened multi-round (strong) unforgeability definition and our TCHF contributions might be of independent interest.

We instantiate TCHF under the DDH assumption in prime-order groups. As a result, we can lift the security guarantees of any weakly-secure threshold signature under standard assumptions. The lifted threshold signature requires only two additional message-independent rounds (which can be preprocessed) and a single additional online round. The overhead in signature size is small: signatures consist of a signature of the to-be-lifted scheme and the randomness of the chameleon hash function. Given our instantiation, this amounts to 2 additional $\mathbb{Z}_p$ elements. Combining our TCHF constructions with, *e.g.*, the construction by Mitrokotsa et al. [MMS+24], yields, to the best of our knowledge, the first strongly-unforgeable threshold signatures in the standard model from standard assumptions (without generic MPC). Similarly, we obtain the first threshold signature that satisfies the highest level of security in the interactive analogy of Bellare et al. [BCK+22]'s hierarchy in the standard model.

On a technical level, we obtain our transformation by modifying and translating the transformation by Steinfeld, Pieprzyk and Wang [SPW07] to the threshold setting, which combines a signature with chameleon hash functions [KR00] to obtain strong unforgeability (cf. fig. 1). Interestingly, we not only boost non-strong unforgeability to strong unforgeability, but also the level in Bellare et al. [BCK+22]'s hierarchy.

1.2 Related Work

Before we detail our techniques, we discuss additional related work.

Chameleon Hash Functions. Bellare and Ristov [BR08] show that Σ-protocols yield chameleon hash functions (CHFs). It is also well-known that Σ-protocols yield signatures via the Fiat-Shamir transformation [FS87]. Looking ahead, we leverage this relation to construct our threshold chameleon hash function.

Camenisch et al. [CDK+17] introduces CHFs with ephemeral trapdoors. In such schemes, there is a master and ephemeral trapdoor for collision-finding. While the master trapdoor is generated during setup of the CHF, the ephemeral

trapdoor is generated during hash evaluation. In order to find the collision, both the master and the ephemeral trapdoor are required. This property and its policy-based extension [DSSS19] are, *e.g.*, useful for sanitizable signatures.

Perhaps the closest work to our threshold CHF definition is [AMVA17]. Ateniese et al. [AMVA17] provide a distributed protocol for collision-finding and provide a UC-based security notion. This notion is, *e.g.*, useful for controlled edits in blockchains. In contrast, our notion is game-based and heavily relies on setting up the image point y in a distributed manner. Further, we require additional properties such as preimage resistance and image unbiasability for our framework. We refer to section 2 for details.

Security Properties of Threshold Signatures. The aforementioned security hierarchies [BCK+22, LNÖ25] and strong unforegeability [NT24] notions for threshold signatures are in the selective-corruptions setting, *i.e.*, the adversary chooses the set of corrupted users *before* interacting with the signing oracles. In contrast, several works consider *adaptive* unforgeability where the adversary can corrupt up to $T-1$ signers at any time [LJY14, BL22, BLT+24, CKM23, KRT24, DR24, Che25]. As prior definitional frameworks for stronger threshold signature notions are in the selective corruption model, we focus on this setting in our work. We believe that, assuming the to-be-lifted threshold signature is adaptively-secure, we can obtain an adaptive boosting framework with a TCHF notion with adaptive corruptions. We believe that it is likely that, *e.g.*, techniques from [KRT24], combined with our construction, yield such an TCHF. As this is out of scope, we leave it for future work.

2 Technical Overview

Before we describe our framework, let us briefly discuss our unforgeability definitions for threshold signatures.

Assumed security notion. We aim to upgrade the security of signature schemes that only satisfy the weakest unforgeability notion in [BCK+22]'s hierarchy (extended to interactive signing protocol). This notion declares a forgery as trivial if the associated message m has been queried to any signing oracle. This means that a signing session on m might have been started and later aborted, yet any forgery on m is classified as trivial.

Our achieved security notion. We aim to achieve the strongest notion, where the adversary A is tasked to output ℓ signatures $(\sigma_i)_{i\in[\ell]}$ on some message m. The forgeries are declared as trivial only if the adversary completed (at least) ℓ *entire* signing sessions for the message m. We model this intuitive classification by adopting the post-fix session identification via sub-session identifiers from [BGHJ24, LNÖ25]. That is, we consider a session on message m completed if all honest signers completed the last round of the signing protocol on message m and agree on the set of honest co-signers $\mathcal{S}_{\mathcal{H}} \subseteq \mathcal{S}$, some arbitrary context ctx and the sub-session identifier ssid. Note that ssid is output in the last signing round by each signer and, *e.g.*, specifies the signer's view during the signing transcript. Observe that ssid plays the role of the leader request in [BCK+22].

2.1 The CHF-based Framework

Since the transformation by Steinfeld, Pieprzyk and Wang [SPW07] serves as the starting point of our work, let us give a brief overview. Chameleon hash functions (CHF) [KR00] are (randomized) collision-resistant hash functions for which there exists a trapdoor that allows to efficiently find a second preimage for any input/output pair [4]. To boost an EUF-CMA secure signature scheme to strong EUF-CMA security (in the non-distributed setting), the idea of Steinfeld, Pieprzyk and Wang [SPW07] is to make the signature σ "dependent on itself". They achieve such circular property by utilizing chameleon hash functions. The signing procedure is as follows:

- Instead of signing the actual message m, sign an output $y := \mathsf{H}(x', r')$ of a collision resistant chameleon hash function H (for fixed x' and random r'). Let $\tilde{\sigma}$ be the signature on y.
- Then, employ the trapdoor for H to find an r that maps the pair of signature $\tilde{\sigma}$ and message m back to y, *i.e.*, $\mathsf{H}(\tilde{\sigma} \| m, r) = y$.
- As the final signature σ, output both $\tilde{\sigma}$ and r.

For verification, given a signature $\sigma = (\tilde{\sigma}, r)$ for a message m, first compute $y := \mathsf{H}(\tilde{\sigma} \| m, r)$ and then verify that $\tilde{\sigma}$ is a valid signature for y. Indeed, given a set of message/signature pairs $(m_i, \sigma_i)_{i \in \mathcal{I}}$, assuming collision resistance of H and (non-strong) unforgeability of the base signature scheme it is hard to find a forgery (m^*, σ^*) with $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all $i \in \mathcal{I}$, as required to break strong unforgeability.

The problem with this simple construction is, that a security reduction proving strong unforgebility needs to create valid message/signature pairs (m_i, σ_i) in the first place, *i.e.*, it needs to simulate the signing oracle in the security game for unforgeability, thereby breaking circularity and finding a signature for the hash of the signature. This can only be done by knowing *both* the secret signing key and the trapdoor for the chameleon hash function.

Hence, in the final construction, a second chameleon hash function F is chained in between the evaluation of H and the signing procedure as shown in the left part of fig. 1. This allows to break circularity through either of the two chameleon hash functions F and H, *i.e.*, for signing, knowledge of a trapdoor for one of the two hash functions is sufficient. A signature now consists of both random strings r_{H} and r_{F} as well as the signature $\tilde{\sigma}$ (shown in orange in the figure), and the secret key consists of the secret key sk of the base signature scheme and the trapdoor for H. The security reduction based on collision resistance of the two chameleon hash functions H and F, and unforgeability of the signature scheme then works by sampling keys for two of the primitives when aiming to break security of the third, respectively.

[4] For some CHF H, a preimage to $y = \mathsf{H}(x, r)$ consists of the input x and the randomness r. We sometimes use the term preimage for just the randomness r if the input x is clear by context.

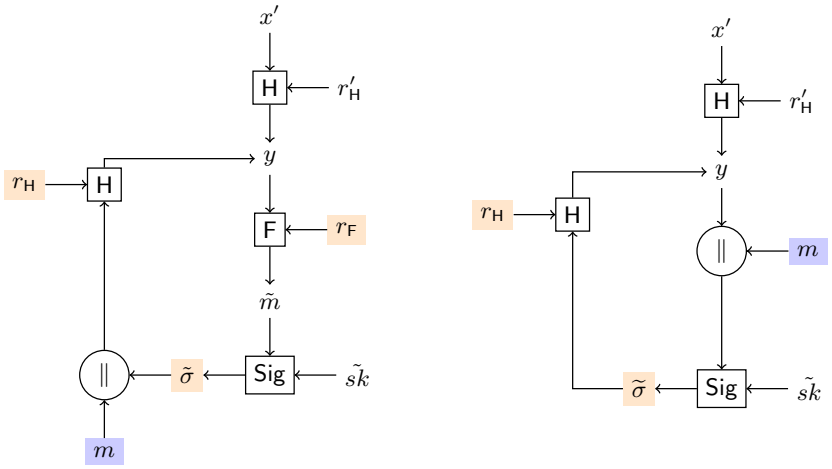

Fig. 1. Relation between message and signature values in the original framework by [SPW07] (left) and our modified framework (right) assuming adaptive collision-resistance which is easier to thresholdize. Blue value represents the input message m and orange values the signature on m.

2.2 A Simplifcation of the Framework

To explain our distributed framework, let us first introduce a conceptual simplification of [SPW07]'s framework in the single user case. [5] In particular, we show that a stronger security notion for CHF allows to prove security of the framework sketched above *without* the second chameleon hash function F. Looking ahead, a construction with just one chameleon hash function is much simpler to thresholdize, and hence this conceptual simplification allows us to provide a round-efficient threshold framework with strong guarantees.

Before we explain the stronger security notion (adaptive collision-resistance) of H and how it helps in the proof, let us mention that we make one additional change to the framework. Namely, we change the place in which m is introduced: instead of signing y, we sign $y\|m$ to obtain $\tilde{\sigma}$, and then employ the trapdoor of the chameleon hash function H to close the circle (cf. right side of fig. 1).[6] This will simplify the security proof.

Adaptive collision-resistance. Our simplification is based on the following observation. While the reduction must simulate the signing oracles, the oracle does *not* need to be able to compute collisions itself. That is, only the value r_{H} computed via the trapdoor is output as part of the signature, but not r'_{H}. In particular, the circle can be closed if the game can sample an *adaptively-chosen*

[5] This framework is implicit in our work and obtained by adapting our threshold signature framework to the non-distributed case. We describe it here for exposition.

[6] In our final construction discussed in section 5, we additionally insert a collision-resistant hash function f after the signing algorithm to map $\tilde{\sigma}$ back to the message space of H.

preimage for H for an image point y. If no other preimage is known for y, then this does *not* produce a collision for H. To allow for such simulation, we modify the collision-resistance game of CHF as follows. The adversary has access to two oracles:

1. $\mathcal{O}^{\mathsf{Eval}}(x)$: Outputs an H image y for x *without* revealing the randomness r used to sample y.
2. $\mathcal{O}^{\mathsf{TrapColl}}(y, x^*)$: Given an image y produced by the Eval-oracle and an input x^*, outputs r^* such that $y = \mathsf{H}(x^*, r^*)$. Here, x^* can be chosen *adaptively* after y is determined. This oracle can be invoked at most once per y.

The adversary is still tasked to find a full collision for H.

Such stronger security notion allows to prove strong unforgeability as follows: Assume that the adversary outputs its forgery $m^*, \sigma^* = (\tilde{\sigma}^*, r^*)$. We can compute the image $y^* = \mathsf{H}(\tilde{\sigma}^*, r^*)$ from the signature and distinguish two cases:

Case $y^* \| m^*$ never signed. This immediately breaks (non-strong) unforgeability of the underlying signature.

Case $y^* \| m^*$ signed before. We argue that in this case, we can find an H collision.

To simulate a signature on some message m, the reduction samples an image y via $\mathcal{O}^{\mathsf{Eval}}(x')$ (recall that x' is fixed a-priori). Then, the reduction signs $y \| m$ to obtain $\tilde{\sigma}$ and invokes $\mathcal{O}^{\mathsf{TrapColl}}(y, \tilde{\sigma})$ to obtain r such that $\mathsf{H}(\tilde{\sigma}, r) = y$. It answers the signing query with the signature $\sigma = (\tilde{\sigma}, r)$.

It remains to discuss how the reduction extracts a collision from the forgery (m^*, σ^*). As we assume that $y^* \| m^*$ has been signed before, there is a signing query that yields another preimage $y^* = \mathsf{H}(\tilde{\sigma}, r)$. Furthermore, the tuples $(\tilde{\sigma}^*, r^*)$ and $(\tilde{\sigma}, r)$ must be distinct by the winning condition of strong unforgeability, therefore this breaks adaptive collision-resistance of H.

In section 2.4 we discuss the difficulty of achieving adaptive collision-resistance for CHF from standard assumptions.

2.3 Our Distributed Framework for Threshold Signatures

For our threshold framework, we require a threshold chameleon hash function (TCHF) H that allows T-out-of-N parties holding shares of the trapdoor to produce a preimage for some image y that is sampled beforehand. Both evaluation (to determine the image y) and collision finding (to compute the preimage r) are now distributed protocols DEval and $\mathsf{DTrapColl}$, respectively, with T parties [7]. In a bit more detail, parties first jointly execute DEval to produce an image y. Additionally, each party might keep some private state (e.g., randomness they used during the DEval protocol). Using their trapdoor shares and private states, parties run $\mathsf{DTrapColl}$ to find randomness r such that $y = \mathsf{H}(x, r)$ for some x,

[7] Let us remark that the $\mathsf{DTrapColl}$ protocol does *not* generate a collision, only a preimage (x, r). We keep the name $\mathsf{DTrapColl}$ for consistency with prior works on chameleon hash functions.

potentially chosen *after* the protocol DEval was ran. Observe that again, the protocol itself reveals only the final randomness, so we define adaptive collision-resistance similar to the non-distributed notion. That is, after corrupting at most $T-1$ parties, the adversary should not be able to find a collision, even given access to DEval and DTrapColl protocols. The message x for DTrapColl can be chosen adaptively. Again, it is crucial that only the preimage (x, r) for y is revealed.

Compared to our simplified framework for the single user case, we now replace the non-distributed chameleon hash function and signature with a TCHF and a threshold signature scheme to boost the security of the threshold signature scheme. Looking at the right side of fig. 1, the algorithms computing y, $\tilde{\sigma}$ and r_{H} would be replaced with the respective threshold protocols.

Additional TCHF properties. It turns out that adaptive collision-resistance is *not* sufficient to prove security of the threshold version of our construction. The issue is that in the distributed setting, the adversary can arbitrarily interleave sessions or even abort sessions before their completion. For instance, the adversary might employ the signer oracles to obtain a signature $\tilde{\sigma}$ on $y\|m$ and then produce an appropriate preimage for y without finishing the signing sessions. Under our strong unforgeability notion for threshold signatures, this session does not classify the obtained signature as trivial, therefore the adversary might break unforgeability *without* breaking collision-resistance. Hence, we need two additional security properties:

Preimage resistance. It is hard to find a preimage for an image point y determined by DEval without finishing the session, *i.e.*, the DTrapColl protocol must be executed fully by all honest users.

Image unbiasability. The image y is unique, *i.e.*, y will be the output of DEval at most once amongst all sessions.

We note that we formalize preimage resistance via sub-session identifiers ssid and context ctx, similar to the threshold signature definition, to enforce that parties agree on their public view within the session (without having to introduce pre-fix session identifiers). These additional properties, together with adaptive collision-resistance, allow us to prove strong security guarantees of the resulting threshold signature.

Interestingly, while the natural analogues of these properties in the non-distributed setting are implied by collision-resistance, this is *not* the case in the distributed setting. For instance, the adversary might be able to enforce some specific image y into the DEval protocol to break image unbiasability, without being able to find an appropriate preimage. Indeed, a variant of our TCHF construction in the group setting (which we discuss later) is adaptively collision-resistant but *not* image unbiasable.

Proving stronger unforgeability. The proof of one-more unforgeability of our threshold signature scheme roughly follows the proof outline of the non-distributed construction, however, several technical challenges are introduced due to the distributed nature of the protocol.

Assume that an adversary against the one-more unforgeability outputs a forgery $(m^*, (\sigma_i^*)_{i\in[\ell]})$. From the ℓ signature, we can compute the corresponding images $(y_i^*)_{i\in[\ell]}$. The first case is very similar as in the single user case:

Case 1: $\exists i\colon\ y_i^*\|m^*$ **never signed.** In this case, we assume that for one of the image points y_i^* it holds that no honest party has ever started a signing session for the message $y_i^*\|m^*$.[8] This case can easily be reduced to the unforgeability of the underlying signature scheme.

Case 2: $\exists i \neq j\colon\ y_i^* = y_j^*$**.** In this case, we assume that we have a collision in the image values derived from the ℓ forged signatures. It is not difficult to show that this leads to a collision of H.

Case 3: none of the above applies. This is the most technical part of our proof. We show that if all $y_i^*\|m^*$ have been signed and all the image points y_i^* are pairwise distinct, then either we break unbiasability or preimage resistance. Intuitively, the image unbiasability property guarantees that each of the ℓ image points y_i^* was derived in exactly one DEval session (with high probability). Since case 1 did not happen, we know that this must have been in an m^* signing session.

By one-more unforgeability, we know that strictly less than ℓ signing sessions for m^* have been completed, yet, the adversary was able to output ℓ valid signatures. Intuitively, this means that the adversary was able to find a preimage for one of the image values y_i^* even though the corresponding $\mathsf{DTrapColl}$ session was not completed. This breaks the preimage resistance of the TCHF H.

2.4 TCHF Instantiation

Before discussing our instantiation of TCHF, we discuss the challenge in constructing even single-user adaptively collision-resistant CHF. Later we explain our solution to this challenge based on standard assumptions and how to thresholdize it.

Achieving adaptive collision-resistance. We argued that adaptive collision-resistance simplifies the framework, but we still need to instantiate such a chameleon hash function H. However, there is a barrier towards achieving this goal. Intuitively, the adaptive choice of the preimage x^* in the $\mathsf{TrapColl}$ enforces that the simulation *can* produce collisions. To see this, observe that the simulation must be able to provide preimages for arbitrary x^* with high probability. As x^* is chosen adaptively, it must, in particular, be able to produce a preimage for distinct x_0^*, x_1^*, breaking collision resistance.

For instance, we believe that this property cannot be achieved in the standard model under a non-interactive assumptions for the original CHF construction by [KR00] which works over a group $\mathbb{G}$ of prime-order p with generator G. Given another generator H, an image Y for x is computed by sampling $r \leftarrow \mathbb{Z}_p$

[8] Recall that we assume the weakest notion of unforgeability for the underlying signature scheme

and setting $Y = xG + rH$. Under the DL assumption, it is easy to see that this CHF is collision-resistant by embedding a DL challenge into H. However, if a reduction is able simulate the $\mathsf{TrapColl}$-oracle for [KR00]'s CHF, then the reduction intuitively can break DL itself:

- To break DL, we forward a fresh DL challenge to the reduction which embeds the challenge into H.
- We can simulate the adversary in the game by calling $\mathcal{O}^{\mathsf{Eval}}(x)$ to obtain some $Y \in \mathbb{G}$.
- Then, we call $\mathcal{O}^{\mathsf{TrapColl}}(Y, x_0)$ for some $x_0 \in \mathbb{Z}_p$ and obtain r_0 such that $Y = x_0G + r_0H$.
- Let $x_1 \neq x_0$ be some $\mathbb{Z}_p$ element. While we cannot invoke $\mathcal{O}^{\mathsf{TrapColl}}(Y, x_1)$ anymore, we can *rewind* the reduction to the point just before the $\mathsf{TrapColl}$-oracle is called. Then, we obtain a second r_1 such that $Y = x_1G + r_1H$.
- Finally, given (x_0, r_0) and (x_1, r_1) with $x_0 \neq x_1$, we can *always* compute the DL of H.

While the above is not a formal argument, we hope it illustrates the challenge.

An adaptive CHF. We base our construction on the chameleon hash function from [KR00]: the image is of the form $Y = xG + rH$ with $H = hG$. To compute a collision for input x^*, we can set $r^* = (x^* - x)\tau + r$ via the trapdoor $\tau = h^{-1} \bmod p$. As described above, it seems that we cannot prove adaptive collision-resistance for this construction. Therefore, we make the following simple but crucial modification:

$$Y = xG + r_0H_0 + r_1H_1.$$

Here, $H_b = h_bG$ is part of the CHF description and the values $r_0, r_1 \leftarrow \mathbb{Z}_p$ form the CHF randomness. As in [KR00], we can employ $h_0^{-1} \bmod p$ to compute a preimage. But now, we can also employ $h_1^{-1} \bmod p$ to do so, and importantly, it is indistinguishable which trapdoor was employed to compute the preimage. Thus, we can simulate adaptively-chosen preimages for images Y in two distinct modes. When the adversary outputs its collision, we can compute either the discrete logarithm of H_0 or H_1, depending on how the collision is distributed. If we play our cards right, then this property allows us to prove adaptive collision-resistance under the DL assumption.

Our Threshold CHF. To build a distributed variant, we can rely on the techniques from threshold Schnorr signatures (*e.g.*, [CKM23]), as the response resembles the structure of Schnorr. That is, each user chooses some $R_i = r_{0,i}H_0 + r_{1,i}H_1$ in DEval, and then outputs $z_{0,i} = (x - x^*)\tau_i + r_{0,i}$ and $z_{1,i} = r_{1,i}$ in $\mathsf{DTrapColl}$, where τ_i are secret shares of $\tau := h_0$.

To prove adaptive collision resistance, our goal is to simulate the DEval and $\mathsf{DTrapColl}$ oracles either with h_0 or with h_1 (such that the adversary cannot distinguish the cases). For Schnorr signatures, a natural simulation strategy relies on partial keys $X_i = \tau_iG$ and simulating the response via honest-verifier zero-knowledge. Adapting this approach to our setting, a valid response $(z_{0,i}, z_{1,i})$ can be simulated by embedding $R_i = (x - x^*)X_i + z_{0,i}G + z_{1,i}H_1$ into the DEval session. The core difference is that for distributed Schnorr signatures, the secret

is not inverted modulo p: this allows to simulate the partial public keys X_i given $X = xG$ via the properties of Shamir's secret sharing, as x_i is a sharing of x and not of $x^{-1} \bmod p$. In our case, this modular inversion makes it difficult to setup the partial keys X_i without knowing the discrete logarithm in the first place.

Instead, we rely on the recent masking technique from [DKM+24]: [9] in our construction, the response $z_{0,i}, z_{1,i}$ is additionally masked with a zero shares $\Delta_{0,i}, \Delta_{1,i}$, respectively. Then, each response $z_{b,i}$ is distributed uniformly over $\mathbb{Z}_p$ conditioned on summing to a correct preimage for Y. Together with the above observations, this allows us to prove adaptive collision-resistance of our TCHF.

At this point, the distributed construction is not yet secure. In particular, the adversary can bias the image Y by choosing its contribution R_c for some corrupted party c to DEval based on the honest contributions R_i. To resolve this, we employ a commit-and-open protocol to determine Y, *i.e.*, each party commits to R_i in a first DEval round and then opens the commitment to R_i in the second round. Given this change, we can also prove *preimage resistance* and *image unbiasability*. The latter crucially requires that the views of all honest users are consistent in every session, which is ensured by signing the round message at the end of DEval, and forcing $\mathsf{DTrapColl}$ and $\mathsf{AggEval}$ to abort if a signature does not pass verification.

In addition to helping with collision-resistance, the properties of $\mathsf{ZeroShare}$ enable us to prove rather strong preimage resistance properties of our construction which we require for our generic framework. In hindsight, similar masking-related arguments are also employed in [LNÖ25, Section 6.2] to achieve higher levels of security, however, their boosting technique is insufficient for our purpose.

Finally, as we wish to avoid the random oracle which is usually employed to realize commit-and-open protocols, we assume that the commitment scheme is both equivocal and extractable. Such commitments are, *e.g.*, instantiatable under DDH. We refer to section 6 for more details.

3 Preliminaries

3.1 Notation

For a function $f : \mathbb{N} \to \mathbb{N}$ we write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to denote that f is a negligible function. For a list S, by writing $\mathsf{S}[\cdot] := x$, we mean the assignment of $\mathsf{S}[i] := x$ for every i. We write ε to denote the empty string and $\perp$ as a special symbol denoting that a variable is undefined, i.e. $X := \perp$ means that the value of X is undefined. We assume every algorithm and oracle outputs $\perp$ when run on an input containing a variable that is set to $\perp$. For an event E we write $[\![E]\!] = 1$ if E is true, and $[\![E]\!] = 0$ else. For a set $\mathcal{R}$, we write $r \leftarrow \mathcal{R}$ to denote that r is sampled uniformly at random from set $\mathcal{R}$.

[9] Note that masking was also used for distributed Schnorr signatures, but for a different purpose than ours [KRT24].

Protocol notation and assumptions. We rely on trusted setup and key generation, consistent with prior game-based definitions for threshold signatures [BCK+22, NT24]. We use natural numbers to refer to parties participating in a protocol.

Consider a protocol run among parties from set $\mathcal{S} \subset \mathbb{N}$ running in R rounds. At the beginning of each round $r \in [R]$, a party $k \in \mathcal{S}$ receives messages that other parties sent to k in round $r-1$ (if $r > 1$). Party k then performs some local computation (which potentially updates the party's private state), and sends messages to other parties in the protocol. We define the local computation of a party k in a given round r as an algorithm that takes as input the party's *state* st_k, a vector of *round messages* $(\mathsf{rm}_{s,r-1})_{s\in\mathcal{S}}$ and potentially some additional inputs. The algorithm outputs an updated (private) state st_k and a round message $\mathsf{rm}_{k,r}$. The round message represents the message being sent from k to all other parties in $\mathcal{S}$ in round r.

Throughout this paper, whenever we refer to a cryptographic primitive, we will assume it to be correct without explicitly mentioning correctness, e.g., by "commitment scheme" we refer to a *correct* commitment scheme.

We recall some standard preliminaries in section A, including collision resistant hash functions, (non-distributed) signatures, PRFs, and commitment schemes.

3.2 Threshold signatures

In the following we provide the definition of (strongly) unforgeable threshold signatures (TS).

We follow the work on threshold (blind) signatures [LNÖ25] and add a context ctx as input into the threshold signing algorithm. As the name suggest, ctx is a string representing values coming from a larger protocol using the TS scheme as a building block. Moreover, we assume that users participating in the signing protocol output a *subsession identifier* ssid in the last round of the protocol. The purpose of this ssid is post-facto identification of a session users believe to have participated in. Looking ahead, this is an important concept in our definition of strong unforgeability, as it allows us to argue whether certain users completed the same signing session. The definition does not dictate how exactly the subsession identifier should be set in the protocol, but it implies that ssid must be of high min-entropy such that two different sessions result in two different ssids (with high probability) and consistent within one session for honest users. We elaborate on these two properties in a bit more detail in remark 1 stated after the formal definition of TS.

Definition 1 (Threshold signatures). *An R-round threshold signature scheme* TS *consists of four algorithms* Setup, KeyGen, Sign, Vrfy *that have the following syntax:*

- $\mathsf{Setup}(1^\lambda, N, T)$*: Takes as input a security parameter (in unary), a total number of users N and a threshold T, and outputs public parameters* pp*, which include λ, N, T, as well as a message space $\mathcal{M}$.*

– $\mathsf{KeyGen}(\mathsf{pp})$*: Takes as input public parameters* pp *and outputs a public key* pk *and secret key shares* $(\mathsf{sk}_i)_{i\in N}$*. The public key* pk *includes the public parameters* pp *and each secret key share* sk_i *includes* pk*.*

– $\mathsf{Sign} = ((\mathsf{Sign}_j)_{j\in[R]}, \mathsf{SigAgg})$*: This is an interactive protocol among a set of users* $\mathcal{S} \subseteq [N]$ *where each participating user* $k \in \mathcal{S}$ *gets as input the set of users* $\mathcal{S}$ *and a message* $m \in \mathcal{M}$*, as well as a secret key share* sk_k *and a context* ctx*. In addition to the public output of a signature* σ*, each user also outputs a subsession identifier* ssid *identifying the session they believe to have participated in.*

- $\mathsf{Sign}_j((\mathsf{rm}_{i,j-1})_{i\in\mathcal{S}}, \mathsf{st}_k)$*: This is an algorithm intended to be run by a party* $k \in \mathcal{S}$ *in round* $j \in [R]$*. It takes as input round messages* $\mathsf{rm}_{i,j-1}$ *of all users* $i \in \mathcal{S}$ *from the previous round (with* $\mathsf{rm}_{i,0} = \varepsilon$*) and state* st_k *(with* $\mathsf{st}_k := (k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{ctx})$ *for* $j = 1$*). It outputs a round message* $\mathsf{rm}_{k,j}$ *and an updated state* st_k*. For* $j = R$*, any private state is deleted and the state* st_k *is set to* ssid*.*
- $\mathsf{SigAgg}(\mathsf{pk}, \mathcal{S}, m, (\mathsf{rm}_{i,r})_{i\in\mathcal{S}, r\in[R]})$*: Takes as input a public key* pk*, a set* $\mathcal{S}$ *of participating users, the message* m*, and a protocol transcript* $(\mathsf{rm}_{i,r})_{i\in\mathcal{S},r\in[R]}$ *and outputs a signature* σ*.*

– $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$*: Takes as input a public key* pk*, a message* $m \in \mathcal{M}$*, and a signature* σ*, and outputs a bit* b*.*

Definition 2 (Correctness). *An* R*-round threshold signature* $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *satisfies* correctness *if for all* $N, T \in \mathsf{poly}(\lambda)$ *with* $0 < T \leq N$*,* $\mathcal{S} \subseteq [N]$ *with* $|\mathcal{S}| = T$*,* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N, T)$*,* $(\mathsf{pk}, (\mathsf{sk}_k)_{k\in N}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*,* $m \in \mathcal{M}$ *and* $\mathsf{ctx} \in \{0,1\}^*$ *it holds*

$$\Pr[\mathsf{Game}^{\mathsf{s-cor}}_{\mathsf{TS}}(1^\lambda, N, T, \mathcal{S}, \mathsf{pk}, (\mathsf{sk}_k)_{k\in\mathcal{S}}, m, \mathsf{ctx}) = 0] \leq \mathsf{negl}(\lambda)$$

where the game $\mathsf{Game}^{\mathsf{s-cor}}_{\mathsf{TS}}$ *is defined as:*

$\underline{\mathsf{Game}^{\mathsf{s-cor}}_{\mathsf{TS}}(1^\lambda, N, T, \mathcal{S}, \mathsf{pk}, (\mathsf{sk}_k)_{k\in\mathcal{S}}, m, \mathsf{ctx})}$

1. **for** $k \in \mathcal{S}$*:* $\mathsf{st}_k := (k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{ctx})$*,* $\mathsf{rm}_{k,0} := \varepsilon$
2. **for** $j \in [R]$*:*
3. **for** $k \in \mathcal{S}$*:* $(\mathsf{rm}_{k,j}, \mathsf{st}_k) \leftarrow \mathsf{Sign}_j((\mathsf{rm}_{i,j-1})_{i\in\mathcal{S}}, \mathsf{st}_k)$
4. $\sigma \leftarrow \mathsf{SigAgg}(\mathsf{pk}, \mathcal{S}, m, (\mathsf{rm}_{i,j})_{i\in\mathcal{S}, j\in[R]})$
5. **output** $[\![\mathsf{Vrfy}(\mathsf{pk}, m, \sigma) = 1]\!]$

In the following, we define standard as well as strong unforgeability. Standard unforgeability guarantees that it is hard to come up with a signature for a message for which *no signing session* has ever been *initiated*. We call this notion *standard*, as it is the security notion that was used in most recent works on threshold signature schemes (*e.g.*, [CKM23, BLT+24]).

Our notion of *strong* unforgeability strengthens security guarantees in two aspects: It guarantees that it is hard to come up with *one more* signature for

a message than the number of signing sessions that have been *completed* for that message. How to count the number of completed sessions in a distributed protocol, however, is non-trivial. Recently, in [LNÖ25], Lehmann et al. proposed a hierarchy of security notions for threshold blind signatures. Our notion of strong unforgeability for (non-blind) threshold signatures is analogous to their strongest security notion. In a nutshell, in order to increase the counter keeping track of the number of completed signatures for a message m, we require that *all honest* users from a signer set completed the last round of an m signing session by outputting the same subsession identifier ssid.

Our strong unforgeability notion is slightly stronger than the recent definition by Navot and Tessaro [NT24], where a counter for a message m is increased as soon as *one honest* user completes the last signing round.

Definition 3 ((Standard/Strong) unforgeability for threshold signatures). *An R-round threshold signature $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ satisfies strong (one-more) unforgeability if for all $N, T \in \mathsf{poly}(\lambda)$ with $0 < T \leq N$ and any PPT adversary A, we have*

$$\mathsf{Adv}^{\mathsf{suf}}_{\mathsf{TS},\mathsf{A}}(1^\lambda, N, T) := \Pr[\mathsf{Game}^{\mathsf{suf}}_{\mathsf{TS},\mathsf{A}}(1^\lambda, N, T) = 1] \leq \mathsf{negl}(\lambda)$$

where the game $\mathsf{Game}^{\mathsf{suf}}_{\mathsf{TS},\mathsf{A}}$ is defined as below, including the gray-shaded steps and excluding the dashed red-shaded step. An R-round threshold signature $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ satisfies (standard) unforgeability if the same is true for parameter ℓ set to 1 in $\mathsf{Game}^{\mathsf{suf}}_{\mathsf{TS},\mathsf{A}}$, excluding the gray-shaded steps and including the dashed red-shaded step.

$\underline{\mathsf{Game}^{\mathsf{suf}}_{\mathsf{TS},\mathsf{A}}(1^\lambda, N, T)}$

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N, T)$
2. $\mathcal{C} \leftarrow \mathsf{A}(1^\lambda, N, T)$ // *set of corrupted users* $\mathcal{C} \subseteq [N]$, $|\mathcal{C}| < T$
3. $(\mathsf{pk}, (\mathsf{sk}_i)_{i\in[N]}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
4. $\mathsf{Q}[\cdot] := \emptyset, \mathsf{S}[\cdot] := \bot, \mathsf{M}[\cdot] := \bot, \mathsf{R}[\cdot] := 0, \mathsf{RM}[\cdot] := \bot, \mathsf{St}[\cdot] := \bot, \mathsf{CTX} := \bot$
5. $\mathsf{ctr}_{\mathsf{sid}} = 0$, $\mathsf{SSID} = \emptyset$
6. $(m^*, (\sigma^*_j)_{j\in[\ell]}) \leftarrow \mathsf{A}^{\mathcal{O}^{\mathsf{NextSes}}, (\mathcal{O}^{\mathsf{Sign}_j})_{j\in[R]}}(\mathsf{pk}, (\mathsf{sk}_j)_{j\in\mathcal{C}})$
7. **output**
   $[\![\ell > |\mathsf{Q}[m^*]| \ \wedge\ \forall i \neq j : \sigma^*_i \neq \sigma^*_j \ \wedge\ \forall j \in [\ell] : \mathsf{Vrfy}(\mathsf{pk}, m^*, \sigma^*_j) = 1]\!]$

$\underline{\mathcal{O}^{\mathsf{NextSes}}()}$

1. $\mathsf{ctr}_{\mathsf{sid}} := \mathsf{ctr}_{\mathsf{sid}} + 1$
2. **output** $\mathsf{sid} = \mathsf{ctr}_{\mathsf{sid}}$

$\underline{\mathcal{O}^{\mathsf{Sign}_j}(\mathsf{sid}, k, \mathcal{S}, m, \mathsf{ctx}, (\mathsf{rm}_i)_{i\in\mathcal{S}})}$ // $\mathsf{sid} \in [\mathsf{ctr}_{\mathsf{sid}}]$, $k \in \mathcal{S} \setminus \mathcal{C}$, $\mathsf{R}[\mathsf{sid}, k] = j - 1$

if $j = 1$: $\forall i \in \mathcal{S} : \mathsf{rm}_i := \varepsilon$

if $j > 1$: $\mathsf{RM}[\mathsf{sid}, k, j-1] = \mathsf{rm}_k, \mathcal{S} = \mathsf{S}[\mathsf{sid}, k], m = \mathsf{M}[\mathsf{sid}, k], \mathsf{ctx} = \mathsf{CTX}[\mathsf{sid}, k]$

1. **if** $j = 1$ **then**
   - $\mathsf{S}[\mathsf{sid}, k] := \mathcal{S}$, $\mathsf{M}[\mathsf{sid}, k] := m$, $\mathsf{CTX}[\mathsf{sid}, k] := \mathsf{ctx}$
   - $\mathsf{St}[\mathsf{sid}, k] := (k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{ctx})$
   - $\mathsf{Q}[m] := \mathsf{Q}[m] \cup \{\mathsf{sid}\}$
2. $(\mathsf{rm}, \mathsf{st}) \leftarrow \mathsf{Sign}_j((\mathsf{rm}_i)_{i \in \mathcal{S}}, \mathsf{St}[\mathsf{sid}, k])$
3. $\mathsf{RM}[\mathsf{sid}, k, j] := \mathsf{rm}, \mathsf{St}[\mathsf{sid}, k] := \mathsf{st}, \mathsf{R}[\mathsf{sid}, k] = j$
4.
   if $j = R$, **then**
   - $\mathsf{ssid} := \mathsf{st}$, $\mathcal{S}_{\mathcal{H}} := \mathcal{S} \setminus \mathcal{C}$
   - $\mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, m] := \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, m] \cup \{k\}$
   - **if** $\mathcal{S}_{\mathcal{H}} \subseteq \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, m]$, **then** $\mathsf{Q}[m] := \mathsf{Q}[m] \cup \{\mathsf{ssid}\}$
5. **output** rm

Remark 1 (Properties of ssid*).* Note that we did not explicitly state any consistency or uniqueness requirements on ssid. This is because these properties (to the extend needed) are *implied* by the strong one-more unforgeability.

Consistency of ssid**:** Intuitively, if a signing session is successfully completed (meaning that all honest parties completed the last round of signing and the adversary is able to aggregate transcripts into a valid signature), all honest parties should output the same ssid. We argue that such consistency property follows from the strong one-more unforgeability. Assume for contradiction that honest parties output different ssid's in the same signing session of a message m, and an adversary A aggregate to a valid signature on m. Due to the inconsistent ssids, the completed-session counter for m is not increased. Hence, the signature aggregated by A is a valid forgery for m.

Uniqueness of ssid**:** The intuitive property that one would want from ssid is uniqueness across different signing sessions (for the same message m and the same honest user signer set $\mathcal{S}_{\mathcal{H}}$). Again, this property is implied by the strong one-more unforgeability. Assume there are two completed signing sessions for the same message m, context ctx and $\mathcal{S}_{\mathcal{H}}$ and an adversary A is able to aggregate transcripts of both sessions into (different) valid signatures. Moreover, assume that in both sessions, honest users output the same ssid. Then the completed-session counter for m is increased only by one although A is able to output two different valid signatures. Hence, the strong one-more unforgeability is broken.

Remark 2 (Context (in)dependency). Our definition explicitly requires that within one signing session, all honest users run with respect to the same context ctx, i.e., we define one-more unforgeability for schemes that are context-dependant.

One could state an alternative definition in which ctx is not part of the indexing of SSID (i.e., we would have $\mathsf{SSID}[\mathsf{ssid}, \mathcal{S}_{\mathcal{H}}, m]$ instead of $\mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, m]$). Context-dependent schemes would then be captured by such definition through an additional requirement of ssid containing ctx.

4 Definition of Threshold Chameleon Hash Functions

In this section, we define our novel notion of threshold chameleon hash functions (TCHF) – a distributed version of chameleon hash functions, which we recall in the appendix A.5 for reference.

More concretely, T out of N parties can jointly execute a distributed evaluation protocol DEval. As a result, each party holds a public hash value y and their secret state is updated. For any input x^*, parties can engage in a distributed DTrapColl protocol to find some randomness r^* such that (x^*, r^*) hashes to y. To this end, each party uses their secret state and their *share* of the trapdoor.

Following the TS definition, we add a *context* ctx as an additional input to our TCHF protocol and require users to output a post-facto *subsession identifier* ssid in the last round of the protocol execution. Concretely, ctx is provided as an additional input to the distributed DTrapColl protocol, capturing the fact that the final subsession identifier ssid might depend on values computed *between* the execution of DEval and DTrapColl if TCHF is run as a building block in a larger protocol. Looking ahead, this is exactly what happens in our generic construction of the strongly unforgeable threshold signature scheme presented in section 5. We only add ctx to DTrapColl and not to DEval, as we envision the DEval procedure to be run independently of the rest of the larger protocol (as is the case for our construction), potentially in a pre-processing phase.

Definition 4 (Threshold chameleon hash function). *An* (R_e, R_t)*-round* threshold chameleon hash function TCH *consists of five algorithms* Setup, Gen, Eval, DEval, DTrapColl *that have the following syntax:*

- $\mathsf{Setup}(1^\lambda, N, T)$*: Takes as input a security parameter (in unary), a total number of users* N *and a threshold* T*, and outputs public parameters* pp *(which include* λ*,* N*, and* T*).*
- $\mathsf{Gen}(\mathsf{pp})$*: Takes as input public parameters* pp *and outputs the description* ch *of a function* $\mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ *(which includes* pp*) as well as trapdoor shares* $(\tau_i)_{i\in[N]}$ *for* ch*.*
- $\mathsf{Eval}(\mathsf{ch}, x, r)$*: Takes as input the description of a chameleon hash function* ch*, an input* $x \in \mathcal{X}$ *and random coins* $r \in \mathcal{R}$*, and outputs some* $y \in \mathcal{Y}$*.*
- $\mathsf{DEval} = ((\mathsf{DEval}_j)_{j\in[R_e]}, \mathsf{AggEval})$*: This is an interactive protocol among a set of users* $\mathcal{S} \subseteq [N]$ *where each participating user* $k \in \mathcal{S}$ *gets as input the description of a chameleon hash function* ch*, an input* $x \in \mathcal{X}$ *and the set of participating users* $\mathcal{S}$*. In addition to some public output* $y \in \mathcal{Y}$*, the users' secret state is updated.*
  - $\mathsf{DEval}_j((\mathsf{erm}_{i,j-1})_{i\in\mathcal{S}}, \mathsf{st}_k)$*: This is a randomised algorithm intended to be run by party* $k \in \mathcal{S}$ *in round* $j \in [R_e]$*. It takes as input messages* $\mathsf{erm}_{i,j-1}$

of all users $i \in \mathcal{S}$ output in the previous round (with $\mathsf{erm}_{i,0} := \varepsilon$), and the user's state st_k (with $\mathsf{st}_k := (k, \mathsf{ch}, \mathcal{S}, x)$ for $j = 1$). It outputs a round message $\mathsf{erm}_{k,j}$ and an updated state st_k.

- $\mathsf{AggEval}(\mathsf{ch}, \mathcal{S}, x, (\mathsf{erm}_{i,j})_{i\in\mathcal{S}, j\in[R_e]})$*: This deterministic algorithm takes as input the description* ch *of a chameleon hash function, a set $\mathcal{S}$ of participating users, input x and the protocol transcript $(\mathsf{erm}_{i,j})_{i\in\mathcal{S}, j\in[R_e]}$, and outputs some $y \in \mathcal{Y}$.*

– $\mathsf{DTrapColl} = ((\mathsf{DTrapColl}_j)_{j\in[R_t]}, \mathsf{AggColl})$*: This is an interactive protocol among a set of users $\mathcal{S} \subseteq [N]$ where each participating user $k \in \mathcal{S}$ gets as input the description of a chameleon hash function* ch*, the set of participating users $\mathcal{S}$, an input $x^* \in \mathcal{X}$, a trapdoor share τ_k and a context* ctx*. In addition to some public output $r^* \in \mathcal{R}$, each user outputs a subsession identifier* ssid *indicating the session they believe to have participated in.*

- $\mathsf{DTrapColl}_j((\mathsf{trm}_{i,j-1})_{i\in\mathcal{S}}, \mathsf{st}_k)$*: This is an algorithm intended to be run by party $k \in \mathcal{S}$ in round $j \in [R_t]$. It takes as input messages $\mathsf{trm}_{i,j-1}$ of all users $i \in \mathcal{S}$ output in the previous round (with $\mathsf{trm}_{i,0} := \mathsf{erm}_{i,R_e}$), as well as user k's state st_k (for $j = 1$, the state st_k consists of k's output state in the* DEval *protocol, the trapdoor share τ_k, a target value $x^* \in \mathcal{X}$, and a context* ctx *, i.e. $\mathsf{st}_k := (\mathsf{st}_k, \tau_k, x^*, \mathsf{ctx})$). The algorithm outputs a round message $\mathsf{trm}_{k,j}$ and an updated state st_k (with $\mathsf{ssid} := \mathsf{st}_k$ if $j = R_t$).*
- $\mathsf{AggColl}(\mathsf{ch}, \mathcal{S}, x^*, (\mathsf{trm}_{i,j})_{i\in\mathcal{S}, j\in[R_t]})$*: Takes as input the description* ch *of a chameleon hash function, a set $\mathcal{S}$ of participating users, and input $x^* \in \mathcal{X}$, and the protocol transcript $(\mathsf{trm}_{i,j})_{i\in\mathcal{S}, j\in[R_t]}$, and outputs $r^* \in \mathcal{R}$.*

Definition 5 (Correctness). *An (R_e, R_t)-round threshold chameleon hash function* $\mathsf{TCH} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{DEval}, \mathsf{DTrapColl})$ *satisfies* correctness *if for all $N, T \in \mathsf{poly}(\lambda)$ with $0 < T \leq N$, $\mathcal{S} \subseteq [N]$ with $|\mathcal{S}| = T$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N, T)$, $(\mathsf{ch}, (\tau_i)_{i\in N}) \leftarrow \mathsf{Gen}(\mathsf{pp})$, $x, x^* \in \mathcal{X}$, $\mathsf{ctx} \in \{0,1\}^*$ it holds*

$$\Pr[\mathsf{Game}^{\mathsf{ch-cor}}_{\mathsf{TCH}}(1^\lambda, N, T, \mathcal{S}, \mathsf{ch}, (\tau_i)_{i\in\mathcal{S}}, x, x^*, \mathsf{ctx}) = 0] \leq \mathsf{negl}(\lambda)$$

where the game $\mathsf{Game}^{\mathsf{ch-cor}}_{\mathsf{TCH}}$ is defined as

$\underline{\mathsf{Game}^{\mathsf{ch-cor}}_{\mathsf{TCH}}(1^\lambda, N, T, \mathcal{S}, \mathsf{ch}, (\tau_i)_{i\in\mathcal{S}}, x, x^*, \mathsf{ctx})}$

1. **for** $i \in \mathcal{S}$: $\mathsf{st}_i := (i, \mathsf{ch}, \mathcal{S}, x)$, $\mathsf{erm}_{i,0} := \varepsilon$,
2. **for** $j \in [R_e]$:
3. **for** $k \in \mathcal{S}$: $(\mathsf{erm}_{k,j}, \mathsf{st}_k) \leftarrow \mathsf{DEval}_j((\mathsf{erm}_{i,j-1})_{i\in\mathcal{S}}, \mathsf{st}_k)$
4. $y \leftarrow \mathsf{AggEval}(\mathsf{ch}, \mathcal{S}, x, (\mathsf{erm}_{i,j})_{i\in\mathcal{S}, j\in R_e})$
5. **for** $i \in \mathcal{S}$: $\mathsf{trm}_{i,0} := \mathsf{erm}_{i,R_e}$, $\mathsf{st}_i := (\mathsf{st}_i, \tau_i, x^*, \mathsf{ctx})$
6. **for** $j \in [R_t]$:
7. **for** $k \in \mathcal{S}$: $(\mathsf{trm}_{k,j}, \mathsf{st}_k) \leftarrow \mathsf{DTrapColl}_j((\mathsf{trm}_{i,j-1})_{i\in\mathcal{S}}, \mathsf{st}_k)$
8. $r^* \leftarrow \mathsf{AggColl}(\mathsf{ch}, \mathcal{S}, x^*, (\mathsf{trm}_{i,j})_{i\in\mathcal{S}, j\in[R_t]})$
9. **output** $[\![\mathsf{Eval}(\mathsf{ch}, x^*, r^*) = y]\!]$

In the following, we define security for threshold chameleon hash functions. Our notion is selective in the sense that the adversary in the security game has to decide a priory on the set of corrupted users (as is the case for all security notions in the threshold setting considered in this work). However, it is *target-input adaptive* in the sense that the adversary can choose the target value x^* for which it aims to find randomness r^* such that (x^*, r^*) hashes to the image y of DEval *after* the DEval protocol was run. Our security definition consists of three properties: *collision resistance*, *preimage resistance* and *image unbiasability*. Before stating the definition formally, let us provide some intuition about these properties (and why we need all three of them).

It is instructive to first emphasize the main difference between *threshold* chameleon hash functions and their single user counterpart. Recall that the single user evaluation algorithm Eval is a deterministic algorithm that evaluates the chameleon hash function on some input x and it gets its random coins r explicitly as input as well. Hence, the execution of Eval results in full knowledge of a preimage (x, r) for an image y, and a follow-up execution of TrapColl on input x^* provides a second preimage (x^*, r^*) for y. Hence, (Eval, TrapColl) can be used to generate collisions.

In contrast, the protocol DEval is a *randomized* protocol. While DTrapColl might depend on the random coins users used during evaluation DEval, these random coins not necessarily get revealed during protocol execution. In other words, by running DEval on input x, users agree on an image y, but no user actually learns a valid preimage for the image y. Follow-up execution of the distributed DTrapColl protocol on input x^* then provides a preimage (x^*, r^*) for y. But due to the hidden coins of DEval, a collision is not implied. This is also the reason why collision resistance is not enough in our security definition and we need preimage resistance as an explicite additional property.

In a bit more details, let y be an image generated through a DEval protocol execution. Preimage resistance guarantees that unless a DTrapColl protocol finding a preimage for y was completed, it is hard to find a preimage for y. Similarly as in the threshold signature definition, it is not immediately obvious how to formalize that a TCHF session was completed. We chose a very similar approach as in the threshold signature case. Namely, we say that a DTrapColl protocol searching for a preimage of y is completed if all honest users complete the last round of the protocol and output the same subsession identifier ssid.

Finally, let us explain the image unbiasability property and why we need it. At a high level, image unbiasability guarantees that running the protocol DEval twice (even if on the same input x) results in two different output values with high probability. This is formalized by requiring that an honest user never aggregates DEval to the same value y twice, and that DEval sessions with different honest user signing sets never result in the same value y. While intuitively, it might seem that "collisions" in DEval should imply TCHF collisions, this is not necessarily the case. There could be two sessions in which DEval results in the same y, but we are not able to complete the DTrapColl protocol execution in

(one of) these sessions (due to inconsistent views of honest parties, abort etc.) and hence we cannot find two different preimages for y.

The formal definition of TCHF follows.

Definition 6 (Secure Threshold Chameleon Hash Function).

An (R_e, R_t)*-round threshold chameleon hash function* $\mathsf{TCH} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{DEval}, \mathsf{DTrapColl})$ *is* secure *if for all* $N, T \in \mathsf{poly}(\lambda)$ *with* $0 < T \leq N$*, and any PPT adversary* A *it holds*

$$\Pr[\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}(1^\lambda, N, T) = 1] \leq \mathsf{negl}(\lambda)$$

where the game $\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}$ *is defined as*

$\underline{\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}(1^\lambda, N, T)}$

1. $\mathcal{C} \leftarrow \mathsf{A}(1^\lambda, N, T)$ // *set of corrupted users* $\mathcal{C} \subseteq [N]$, $|\mathcal{C}| < T$
2. $\mathsf{ctr}_{\mathsf{sid}} := 0$, $\mathsf{S}[\cdot] := \bot$, $\mathsf{St}[\cdot] := \bot$, $\mathsf{CTX} := \bot$
   // *initialise* sid *counter and lists of user sets, user states and user contexts*
3. $\mathsf{X}_\mathsf{e}[\cdot] = \bot$, $\mathsf{R}_\mathsf{e}[\cdot] := 0$, $\mathsf{RM}_\mathsf{e}[\cdot] := \varepsilon$, $\mathsf{Y}_\mathsf{e}[\cdot] := \bot$ // *initialise lists of inputs, user rounds, user round messages (sent and received) for* DEval*, and user results of* AggEval.
4. $\mathsf{X}_\mathsf{t}[\cdot] := \bot$, $\mathsf{R}_\mathsf{t}[\cdot] := 0$, $\mathsf{RM}_\mathsf{t}[\cdot] := \bot$ // *initialise lists of target inputs, user rounds, user round messages for* DTrapColl
5. $\mathsf{Agg} := \emptyset$, $\mathsf{Cmpl} := \emptyset$, $\mathsf{dCollision} := \mathsf{false}$ // *initialize sets of aggregated and completed values, and a variable for* DEval *collisions*
6. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N, T)$, $(\mathsf{ch}, (\tau_i)_{i \in N}) \leftarrow \mathsf{Gen}(\mathsf{pp})$
7. $(x, r, x^*, r^*) \leftarrow \mathsf{A}^{\mathcal{O}^{\mathsf{NextSes}}, (\mathcal{O}^{\mathsf{DEval}_j})_{j \in [R_e]}, \mathcal{O}^{\mathsf{AggEval}}, (\mathcal{O}^{\mathsf{DTrapColl}_j})_{j \in [R_t]}}(\mathsf{ch}, (\tau_i)_{i \in \mathcal{C}})$
8. **output** 1 *iff one of the three conditions below is satisfied:*

   Collision resistance:
   $\mathsf{Eval}(\mathsf{ch}, x, r) = \mathsf{Eval}(\mathsf{ch}, x^*, r^*) \ \wedge \ (x, r) \neq (x^*, r^*)$
   // A *found a collision*

   Preimage resistance:
   $(y, \cdot, \cdot) \in \mathsf{Agg}$ *and* $y \notin \mathsf{Cmpl}$ *for* $y = \mathsf{Eval}(\mathsf{ch}, x^*, r^*)$
   // A *found a preimage for an incompleted session.*

   Image unbiasability:
   $\mathsf{dCollision} = \mathsf{true}$
   // *Two* DEval *sessions were aggregated to the same value.*

$\underline{\mathcal{O}^{\mathsf{NextSes}}()}$

1. $\mathsf{ctr}_{\mathsf{sid}} := \mathsf{ctr}_{\mathsf{sid}} + 1$
2. **output** $\mathsf{sid} = \mathsf{ctr}_{\mathsf{sid}}$

$\underline{\mathcal{O}^{\mathsf{DEval}_j}(\mathsf{sid}, k, \mathcal{S}, x, (\mathsf{erm}_i)_{i\in\mathcal{S}})}$ // $\mathsf{sid} \in [\mathsf{ctr}_{\mathsf{sid}}]$, $k \in \mathcal{S}\setminus\mathcal{C}$, $|\mathcal{S}| = T$, $\mathsf{R_e}[\mathsf{sid}, k] = j-1$

if $j = 1$: $\forall i \in \mathcal{S}$: $\mathsf{erm}_i = \varepsilon$, $x \in \mathcal{X}$

if $j > 1$: $\mathsf{RM_e}[\mathsf{sid}, k, k, j-1] = \mathsf{erm}_k$, $\mathcal{S} = \mathsf{S}[\mathsf{sid}, k]$, $\mathsf{X_e}[\mathsf{sid}, k] = x$

1. **if** $j = 1$ **then** $\mathsf{S}[\mathsf{sid}, k] := \mathcal{S}$, $\mathsf{St}[\mathsf{sid}, k] := (k, \mathsf{ch}, \mathcal{S}, x)$
2. $\forall i \in \mathsf{S}[\mathsf{sid}, k]$: $\mathsf{RM_e}[\mathsf{sid}, k, i, j-1] := \mathsf{erm}_i$
3. $(\mathsf{erm}, \mathsf{st}) \leftarrow \mathsf{DEval}_j((\mathsf{erm}_i)_{i\in\mathsf{S}[\mathsf{sid},k]}, \mathsf{St}[\mathsf{sid}, k])$
4. $\mathsf{RM_e}[\mathsf{sid}, k, k, j] := \mathsf{erm}$, $\mathsf{St}[\mathsf{sid}, k] := \mathsf{st}$, $\mathsf{R_e}[\mathsf{sid}, k] := j$
5. **output** erm

$\underline{\mathcal{O}^{\mathsf{AggEval}}(\mathsf{sid}, k, (\mathsf{erm}_i)_{i\in\mathsf{S}[\mathsf{sid},k]})}$ // $\mathsf{sid} \in [\mathsf{ctr}_{\mathsf{sid}}]$, $k \in \mathsf{S}[\mathsf{sid}, k]\setminus\mathcal{C}$, $\mathsf{R_e}[\mathsf{sid}, k] = R_e$, $\mathsf{RM_e}[\mathsf{sid}, k, k, R_e] = \mathsf{erm}_k$, $\mathsf{Y_e}[\mathsf{sid}, k] = \perp$

1. $\mathcal{S} := \mathsf{S}[\mathsf{sid}, k]$, $x := \mathsf{X_e}[\mathsf{sid}, k]$
2. $\forall i \in \mathcal{S}$: $\mathsf{RM_e}[\mathsf{sid}, k, i, R_e] := \mathsf{erm}_i$
3. $y \leftarrow \mathsf{AggEval}(\mathsf{ch}, \mathcal{S}, x, (\mathsf{RM_e}[\mathsf{sid}, k, i, j])_{i\in\mathcal{S}, j\in[R_e]})$
4. **if** $y = \perp$ **then output** $\perp$
5. **if** $(y, \cdot, k) \in \mathsf{Agg} \;\vee\; (\exists \mathcal{S}'$ *s.t.* $(y, \mathcal{S}', \cdot) \in \mathsf{Agg}$ *and* $k \notin \mathcal{S}')$ **then** $\mathsf{dCollision} = \mathsf{true}$ *// Earlier* DEval *aggregation to* y *by* k *or with different honest user*
6. $\mathsf{Y_e}[\mathsf{sid}, k] := y$, $\mathsf{Agg} = \mathsf{Agg} \cup \{(y, \mathcal{S}, k)\}$

$\underline{\mathcal{O}^{\mathsf{DTrapColl}_j}(\mathsf{sid}, k, x^*, \mathsf{ctx}, (\mathsf{trm}_i)_{i\in\mathsf{S}[\mathsf{sid},k]})}$ // $\mathsf{sid} \in [\mathsf{ctr}_{\mathsf{sid}}]$, $k \in \mathsf{S}[\mathsf{sid}, k]\setminus\mathcal{C}$, $\mathsf{R_t}[\mathsf{sid}, k] = j-1$, $\mathsf{Y_e}[\mathsf{sid}, k] \neq \perp$

If $j = 1$: $x^* \in \mathcal{X}$, $\forall i \in \mathsf{S}[\mathsf{sid}, k]$: $\mathsf{trm}_i = \mathsf{RM_e}[\mathsf{sid}, k, i, R_e]$

If $j > 1$: $x^* = \mathsf{X_t}[\mathsf{sid}, k]$, $\mathsf{ctx} = \mathsf{CTX}[\mathsf{sid}, k]$, $\mathsf{trm}_k = \mathsf{RM_t}[\mathsf{sid}, k, j-1]$

1. **if** $j = 1$ **then** $\mathsf{X_t}[\mathsf{sid}, k] := x^*$, $\mathsf{CTX}[\mathsf{sid}, k] := \mathsf{ctx}$, $\mathsf{St}[\mathsf{sid}, k] := (\mathsf{St}[\mathsf{sid}, k], \tau_k, x^*, \mathsf{ctx})$
2. $(\mathsf{trm}, \mathsf{st}) \leftarrow \mathsf{DTrapColl}_j((\mathsf{trm}_i)_{i\in\mathsf{S}[\mathsf{sid}]}, \mathsf{St}[\mathsf{sid}, k])$
3. $\mathsf{RM_t}[\mathsf{sid}, k, j] := \mathsf{trm}$, $\mathsf{St}[\mathsf{sid}, k] := \mathsf{st}$, $\mathsf{R_t}[\mathsf{sid}, k] := j$
4. **if** $j = R_t$ **then**
   (a) $\mathsf{ssid} := \mathsf{st}$, $\mathcal{S}_{\mathcal{H}} := \mathsf{S}[\mathsf{sid}, k]\setminus\mathcal{C}$, $y := \mathsf{Y_e}[\mathsf{sid}, k]$, $\mathsf{ctx} := \mathsf{CTX}[\mathsf{sid}, k]$
   (b) $\mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, y] := \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, y] \cup \{k\}$
   (c) **if** $\mathcal{S}_{\mathcal{H}} \subseteq \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, y]$, **then** $\mathsf{Cmpl} := \mathsf{Cmpl} \cup \{y\}$
5. **output** trm

Remark 3 (Properties of ssid*).* Very similarly to the definition of Threshold Signatures (c.f. remark 1), consistency and uniqueness of ssid's is implied by the security properties of TCH (concretely preimage resistance and image unbiasability).

For the proof of our generic construction, it is useful to realize that if a threshold chameleon hash function satisfies image unbiasability, every value y

can be added to Cmpl only once. This is formalized and proven by the following technical lemma.

Lemma 1. *Assume a threshold chameleon hash function satisfying image unbiasability. Then the following implication holds:*

$$y \in \mathsf{Cmpl} \Rightarrow \exists!(\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}})\colon \mathcal{S}_{\mathcal{H}} \subseteq \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, y].$$

Proof. By definition of the set Cmpl, if $y \in \mathsf{Cmpl}$, then there exists $(\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}})$ such that $\mathcal{S}_{\mathcal{H}} \subseteq \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, y]$ and $\forall k \in \mathcal{S}_{\mathcal{H}}$; $\exists \mathcal{S}$ such that $\mathcal{S}_{\mathcal{H}} \subseteq \mathcal{S} \setminus \mathcal{C}$ and $(y, \mathcal{S}, k) \in \mathsf{Agg}$. In other words, all users from the set $\mathcal{S}_{\mathcal{H}}$ aggregated DEval to y, and in that session completed $\mathsf{DTrapColl}$ executed in the context ctx by outputting the subsession identifier ssid. We need to show that the image unbiasability property implies that the triple $(\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}})$ is unique.

The first condition of image unbiasability says that an honest party never aggregates to y twice. Hence, if an honest party $k \in \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, y]$, then $k \notin \mathsf{SSID}[\mathsf{ssid}', \mathsf{ctx}', \mathcal{S}'_{\mathcal{H}}, y]$ for any $(\mathsf{ssid}', \mathsf{ctx}', \mathcal{S}'_{\mathcal{H}}) \neq (\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}})$. This means that if there are two different tuples $(\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}})$ and $(\mathsf{ssid}', \mathsf{ctx}', \mathcal{S}'_{\mathcal{H}})$, the honest user sets $\mathcal{S}_{\mathcal{H}}$ and $\mathcal{S}'_{\mathcal{H}}$ must be disjoint. But this implies that there must exist honest users $k \neq k'$ and user sets $\mathcal{S}, \mathcal{S}'$ with $\mathcal{S}_{\mathcal{H}} = \mathcal{S} \setminus \mathcal{C}$, $\mathcal{S}'_{\mathcal{H}} = \mathcal{S}' \setminus \mathcal{C}$ and $\mathcal{S}_{\mathcal{H}} \cap \mathcal{S}'_{\mathcal{H}} = \emptyset$ such that $(y, \mathcal{S}, k)$, $(y, \mathcal{S}', k') \in \mathsf{Agg}$. Assume $(y, \mathcal{S}', k')$ was added to Agg before $(y, \mathcal{S}, k)$. Since $k \notin \mathcal{S}'$, this gives a contradiction to the second property of image unbiasability. Thus, the triple $(\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}})$ satisfying the property of the claim must be unique.

5 Generic Transformation to Strong Unforgeability

In this section, we describe our generic transformation that turns a (*standard*) *unforgeable* threshold signature scheme into a *strongly unforgeable* threshold signature scheme utilizing the novel distributed primitive of threshold chameleon hash functions introduced in the previous section. We recommend the reader to follow the high-level pseudo-code description of the protocol given in fig. 2 and the right part of fig. 1 while reading the following description. We stress that the distributed signing is described at a high level here and excludes somewhat tedious but important details (i.e., how state is passed between different distributed building blocks). We present our protocol round-by-round in section B.

Let $\tilde{\Sigma}$ be an unforgeable threshold signature scheme and H a collision-resistant threshold chameleon hash function with input space $\mathcal{X}_{\mathsf{H}}$. Moreover, let h be a collision-resistant hash function mapping $\tilde{\Sigma}$ signatures to $\mathcal{X}_{\mathsf{H}}$. We construct a *strongly unforgeable* threshold signature scheme $\Sigma = (\Sigma.\mathsf{Setup}, \Sigma.\mathsf{Gen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Vrfy})$ as follows.

Basic protocol description. The $\Sigma.\mathsf{Setup}$ algorithm simply runs the setup algorithms of the building block primitives and outputs public parameters pp consisting of the public parameters of the building blocks.

The key generation algorithm $\Sigma.\mathsf{Gen}$ on input the public parameters pp first executes the (key) generation algorithms of the building block primitives. Then

it chooses an input value $x' \in \mathcal{X}$ uniformly at random. The output public key pk consists of the public parameters pp, the chosen threshold chameleon hash function ch_H, the public key for $\tilde{\Sigma}$, the chosen hash function f_h and the fixed input value x'. Each secret key share sk_i consists of the public key, a secret key share $\widetilde{\mathsf{sk}}_i$ for $\tilde{\Sigma}$ and a trapdoor share τ_i for ch_H.

The signing algorithm $\Sigma.\mathsf{Sign}$ on input a signer set $\mathcal{S}$, the message m and context ctx, firstly executes the distributed evaluation of the hash function ch_H on input x' (which was fixed during key generation). As a result, each user $i \in \mathcal{S}$ possesses an output y and a private state est_i. Thereafter, users engage in a distributed signing protocol of $\tilde{\Sigma}$, signing the message $y\|m$. After completion of the signing process, users hold the signature $\tilde{\sigma}$. Each party can now locally evaluate f_h on $\tilde{\sigma}$ to obtain x. Finally, parties jointly run the distributed $\mathsf{H.DTrapColl}$ protocol to find r_H such that $\mathsf{H.Eval}(\mathsf{H}, x, r_\mathsf{H}) = y$, thereby closing the signing circle. Importantly, parties initiate $\mathsf{H.DTrapColl}$ with their private state containing their private output state est_i from the $\mathsf{H.DEval}$ protocol. They run the protocol in the context $\mathsf{ctx}_\mathsf{t} = (\mathsf{ctx}, m)$, i.e., the context of the overall signing protocol and the message m being singed. The output of the signing protocol is a signature $\sigma = (\tilde{\sigma}, r_\mathsf{H})$. Recall that by definition the final state of each party defines the subsession identifier ssid of the given party. In our construction ssid is set to be the subsession identifier output by the party in the $\mathsf{H.DTrapColl}$ protocol together with the target output value y.

The verification algorithm $\Sigma.\mathsf{Vrfy}$ is rather straightforward. The algorithm simply parses the signature, recomputes the value x by evaluating f_h on $\tilde{\sigma}$ and computes y by evaluating ch_H on x and randomness r_H. Finally, it verifies that $\tilde{\sigma}$ is a valid signature for the message $y\|m$.

Before stating and proving security of our generic construction, let us make a few remarks.

Remark 4. First, note that the $\mathsf{H.DEval}$ protocol is completely independent of any input parameters (in particular, also the context ctx) except for the set of signers. Hence, this part of the signing protocol can be run in a pre-processing phase, even before the message m to be signed is known. The $\tilde{\Sigma}.\mathsf{Sign}$ protocol, on the other hand, is independent of the TCHF H, i.e. it only relies on the output y of $\mathsf{H.DEval}$ but not on any H-related parameters or the secret state users derive during $\mathsf{H.DEval}$ execution. Finally, the $\mathsf{H.DTrapColl}$ protocol only relates to the TS $\tilde{\Sigma}$ through the hash of the signature $\tilde{\sigma}$ as well as the context ctx_t, that in addition to ctx also contains the message m to be signed. In contrast, the $\mathsf{H.DTrapColl}$ protocol strongly relates to the $\mathsf{H.DEval}$ protocol and in particular all secret user states est_i.

It is easy to see that Σ satisfies correctness if the underlying primitives do. In the remainder of this section, we focus on the security of our generic transformation.

Theorem 1. *If in Construction 2 all building blocks satisfy (the primitive-specific notions of) correctness and*

- *$\tilde{\Sigma}$ is a (standard) unforgeable threshold signature scheme,*

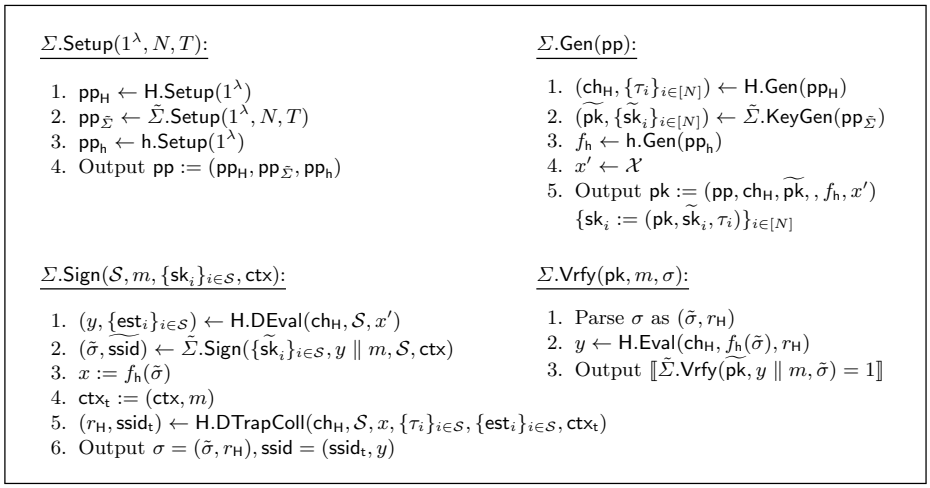

Fig. 2. A simplified description of our generic construction of a strongly unforgeable threshold signature scheme. See Figure 5 for the complete round-by-round formal protocol description.

- H *is a secure threshold chameleon hash function,*
- h *is a collision-resistant hash function,*

then the threshold signature scheme Σ satisfies strong unforgeability.

Proof. Assume A succeeded in providing a valid forgery $(m^*, (\sigma_k^*)_{k\in[\ell]})$ for some parameter ℓ, where all ℓ signatures are pairwise distinct. We can parse the signatures as $\sigma_k^* = (\tilde{\sigma}_k^*, r_{\mathsf{H},k}^*)$, and set $x_k^* := f_\mathsf{h}(\tilde{\sigma}_k^*)$ and $y_k^* := \mathsf{H.Eval}(x_k^*, r_{\mathsf{H},k}^*)$.

We define the following four events:

- E_{forgery} : $\exists i \in [\ell]$ s.t. no $\tilde{\Sigma}$ signing session for $y_i^* \| m^*$ has ever been started in the view of any honest user.
- $E_{\mathsf{col-h}}$: $\exists i, j \in [\ell],\ \tilde{\sigma}_i^* \neq \tilde{\sigma}_j^* \wedge x_i^* = x_j^*$
- $E_{\mathsf{col-H}}$: $\neg E_{\mathsf{col-h}} \wedge \exists i, j \in [\ell],\ i \neq j : y_i^* = y_j^*$
- $E_{\mathsf{preim}} = \neg E_{\mathsf{forgery}} \wedge \neg E_{\mathsf{col-h}} \wedge \neg E_{\mathsf{col-H}}$.

If E_{forgery} happens, we can reduce to unforgeability of the signature scheme $\tilde{\Sigma}$. If event $E_{\mathsf{col-h}}$ occurs, we reduce to collision resistance of h. If event $E_{\mathsf{col-H}}$ occurs, we reduce to collision resistance of H. If none of the three events happen, we reduce to preimage resistance and output unbiasability of H. We prove this in Lemmata 2, 3, 4, 5.

Lemma 2. *Event E_{forgery} occurs with negligible probability due to unforgeability of $\tilde{\Sigma}$.*

Proof. The reduction simulates the game just as an honest challenger does, except for the parameters and keys corresponding to $\tilde{\Sigma}$, for which it embeds the

challenge parameters and public key. It simulates queries to the signing oracles of Σ via the signing oracles of $\tilde{\Sigma}$. The simulation is perfect. If the adversary initiates q signing sessions for Σ, then the reduction initiates at most q signing sessions for $\tilde{\Sigma}$. In the end of the game, since there exists some $i \in [\ell]$ such that $y_i^* \| m^*$ has never been queried to $\tilde{\Sigma}.\mathsf{Sign}$, the reduction forwards $(y_i^* \| m^*, \tilde{\sigma}_i^*)$ as a valid forgery. This breaks unforgeability of $\tilde{\Sigma}$.

Lemma 3. *Event $E_{\mathsf{col-h}}$ occurs with negligible probability under collision resistance of* h.

Proof. The reduction simulates the game just as an honest challenger does, except for the public parameters corresponding to h and the function description, for which it embeds the challenge parameters and challenge function description. It answers signing queries exactly as a challenger does and hence the simulation is perfect. In the end of the game, since there exists some $i, j \in [\ell]$ such that $\tilde{\sigma}_i^* \neq \tilde{\sigma}_j^* \wedge x_i^* = x_j^*$, the reduction forwards $(\tilde{\sigma}_i^*, \tilde{\sigma}_j^*)$ as a collision. This breaks the collision-resistance of h.

Lemma 4. *Event $E_{\mathsf{col-H}}$ occurs with negligible probability under collision resistance of* H.

Proof. The reduction simulates $\Sigma.\mathsf{Setup}$ and $\Sigma.\mathsf{Gen}$ just as an honest challenger does, except for the parameters and function description corresponding to H, for which it embeds the challenge parameters and function description.

To simulate the signing oracles, the reduction uses the $\mathsf{H.DEval}$ and $\mathsf{H.TrapColl}$ oracles provided by the collision resistance challenger for H (cf. definition 6).

- For each fresh signing session, i.e., A calls the $\Sigma.\mathsf{NextSes}$ oracle, make a query to the $\mathsf{H.NextSes}$ oracle.
- For each query the adversary makes to a $\Sigma.\mathsf{Sign}_j$ oracle for a round j that belongs to the block of rounds where $\mathsf{H.DEval}$ is evaluated, use the respective $\mathsf{H.DEval}_j$ oracles to compute the respective output. This simulation is perfect.
- For aggregation just follow the honest protocol.
- Engage in the signing protocol on behalf of honest parties to distributedly compute $\tilde{\sigma}$. This simulation is perfect because, while in an honest execution of the protocol users keep their state from the $\mathsf{H.DEval}$ protocol during $\tilde{\Sigma}.\mathsf{Sign}$, the $\tilde{\Sigma}.\mathsf{Sign}$ part of the protocol only depends on the output of the $\mathsf{H.DEval}$-protocol and not on secret internal state derived during $\mathsf{H.DEval}$ as discussed already in remark 4.
- Perform the evaluation of h according to the honest protocol.
- Simulate the $\mathsf{H.DTrapColl}$ part of signing via the $\mathsf{H.DTrapColl}_j$ oracle. Again, this simulation is perfect, because the $\mathsf{H.DTrapColl}$ rounds in $\Sigma.\mathsf{Sign}$ only depend on $\tilde{\sigma}$, m^* and the $\mathsf{H.DEval}$-dependent parts of each user's state.

We will now argue how a collision for H can be extracted from the forgery given that we assume that $E_{\mathsf{col-H}}$ happened. Recall that the event is defined as

$$\neg E_{\mathsf{col-h}} \ \wedge \exists i, j \in [\ell],\ i \neq j : y_i^* = y_j^*.$$

The condition $\neg E_{\mathsf{col-h}}$ implies that $\forall i, j \in [\ell] : \tilde{\sigma}_i^* = \tilde{\sigma}_j^* \vee x_i^* \neq x_j^*$. Hence, we know that there exist $i, j \in [\ell]$ $i \neq j$ such that either

1. $\tilde{\sigma}_i^* = \tilde{\sigma}_j^* \wedge y_i^* = y_j^*$ or
2. $x_i^* \neq x_j^* \wedge y_i^* = y_j^*$.

In both cases, $(x_i^*, r_{\mathsf{H},i}^*)$ and $(x_j^*, r_{\mathsf{H},j}^*)$ form an H collision. In the latter case, this is immediate to see as $x_i^* \neq x_j^*$. In the former case, this follows from the fact that the forgery of the adversary contains ℓ *pairwise distinct* signatures $\sigma_1^*, \ldots, \sigma_\ell^*$. Recall that the signatures are of the form $\sigma_i^* = (\tilde{\sigma}_i^*, r_{\mathsf{H},i}^*)$. Hence if $\tilde{\sigma}_i^* = \tilde{\sigma}_j^*$, it must hold that $r_{\mathsf{H},i}^* \neq r_{\mathsf{H},j}^*$.

Lemma 5. *Event E_{preim} occurs with negligible probability under preimage resistance and image unbiasability of H.*

Proof. The reduction simulates the game exactly as in lemma 4. We will now argue how a preimage for H can be extracted from a forgery given that we assume that E_{preim} happened. Recall that

$$E_{\mathsf{preim}} = \neg E_{\mathsf{forgery}} \wedge \neg E_{\mathsf{col-h}} \wedge \neg E_{\mathsf{col-H}}.$$

Since $\neg E_{\mathsf{col-h}} \wedge \neg E_{\mathsf{col-H}}$, we know that $y_i^* \neq y_j^*$ for all $i, j \in [\ell], i \neq j$. The condition $\neg E_{\mathsf{forgery}}$ implies for each $i \in [\ell]$, there is a user that started a $\tilde{\Sigma}$ signing session on a message $y_i^* \| m^*$. Hence, this user aggregated the transcript of DEval to y_i^* in an m^* signing session. Consequently, we know that for all $i \in [n]$, $(y_i^*, \cdot, \cdot) \in \mathsf{Agg}$. Our goal is to show that there exists $i \in [\ell]$ such that $y_i^* \notin \mathsf{Cmpl}$ and hence the forgery breaks preimage resistance.

For contradiction, assume that for all $i \in [\ell]$ it holds that $y_i^* \in \mathsf{Cmpl}$. By lemma 1, this implies that for every $i \in [\ell]$ there exists a unique tuple $(\mathsf{ssid}_i, \mathsf{ctx}_i, \mathcal{S}_{\mathcal{H},i})$ such that

$$\mathcal{S}_{\mathcal{H},i} \subseteq \mathsf{SSID}[\mathsf{ssid}_i, \mathsf{ctx}_i, \mathcal{S}_{\mathcal{H},i}, y_i^*]. \tag{1}$$

where SSID is the set from the end of the security game of the CHF security definition, ssid_i is the output of parties after $\mathsf{TrapColl}$, ctx_i is the context in which $\mathsf{TrapColl}$ was run and $\mathcal{S}_{\mathcal{H},i}$ is the set of honest parties participating in the $\mathsf{TrapColl}$ protocol.

By our construction, we know that $\mathsf{ctx}_i = (\mathsf{ctx}_i^{(\Sigma)}, m^*)$, where $\mathsf{ctx}_i^{(\Sigma)}$ is the context in which the whole signature scheme Σ is run. Moreover, we know that honest parties output a subsession identifier $\mathsf{ssid}_i^{(\Sigma)}$ which is equal to the subsession identifier output by the party after $\mathsf{TrapColl}$ and y_i^*, i.e, $\mathsf{ssid}_i^{(\Sigma)} = (\mathsf{ssid}_i, y_i^*)$. The honest signer set $\mathcal{S}_{\mathcal{H},i}^{(\Sigma)}$ in Σ is the same as in the underlying $\mathsf{TrapColl}$ protocol (i.e., $\mathcal{S}_{\mathcal{H},i}^{(\Sigma)} = \mathcal{S}_{\mathcal{H},i}$). Finally, the last round of Σ is the last round of the $\mathsf{TrapColl}$ protocol. Hence, we have

Hence eq. (1) implies that for every $i \in [\ell]$

$$\mathcal{S}_{\mathcal{H},i}^{(\Sigma)} \subseteq \mathsf{SSID}^{(\Sigma)}[\mathsf{ssid}_i^{(\Sigma)}, \mathsf{ctx}_i^{(\Sigma)}, \mathcal{S}_{\mathcal{H},i}^{(\Sigma)}, m^*]$$

and therefore $\mathsf{ssid}_i^{(\Sigma)} \in \mathsf{Q}[m^*]$, where $\mathsf{Q}[m^*]$ denotes the set of ssids for which a signing session fo m^* was completed by the end of the security game, as defined in the security definition of TS. As we assume that all the ℓ target output value y_i^* are pairwise distinct, then so are all the ℓ subsession identifiers $\mathsf{ssid}_i^{(\Sigma)} = (\mathsf{ssid}_i, y_i^*)$. This implies that $|\mathsf{Q}[m^*]| \geq \ell$ which is a contradiction with one-more unforgeability of Σ.

6 Group-based Instantiation

We present our threshold CHF instantiation based on prime-order groups. It is based on the CHF by [KR00], where the images are Pedersen commitments $Y = xG + rH$ of the message. Our modification allows to prove *input-adaptive* collision resistance.

6.1 Preliminaries

Let GenG be an algorithm that on input 1^λ, outputs a description of a group $\mathbb{G}$, together with a generator G and $\mathbb{G}$'s order p. We use additive group notation and denote group elements $G \in \mathbb{G}$ by capital letters. We denote $\mathbb{Z}_p$ elements by lowercase letters.

Definition 7 (DL). *The discrete logarithm (DL) assumption states that for any PPT adversary* A*, it holds for* $(\mathbb{G}, G, p) \leftarrow \mathsf{GenG}(1^\lambda)$ *that*

$$\Pr[\mathsf{A}(G, xG) = x \mid x \leftarrow \mathbb{Z}_p] = \mathsf{negl}(\lambda).$$

We recall T-out-of-N Shamir's secret sharing over $\mathbb{Z}_p$ [Sha79]. Let $s \in \mathbb{Z}_p$ denote the to-be-shared secret. Then, $\mathsf{SShare}_{N,T,\mathbb{Z}_p}(s)$ proceeds as follows on input x:

1. sample a random degree $T-1$ polynomial of $P \in \mathbb{Z}_p[X]$ such that $P(0) = x$,
2. output shares $(x_i)_{i\in[N]}$, where $x_i := P(i)$.

We denote the lagrange coefficients by $L_{\mathcal{S},i} = \prod_{j\in\mathcal{S}\setminus\{i\}} \frac{-j}{i-j}$. Let $\mathcal{S} \subseteq [N]$ be an arbitrary subset of size at least T. Note that the s can be reconstructed via

$$s = \sum_{i\in[\mathcal{S}]} L_{\mathcal{S},i} \cdot s_i.$$

6.2 Threshold Chameleon Hash

We introduce our threshold chameleon hash function based on the Discrete Logarithm (DL) assumption.

Let COM be an equivocable and extractable commitment scheme. Let SIG be a EUF-CMA secure signature scheme. Let PRF be a pseudorandom function that maps from $\{0,1\}^{\ell+1}$ into $\mathbb{Z}_p$ where ℓ is the bit-size required to represent

ssid and ctx in our construction (cf. fig. 3). Let $\mathbf{seed}_i = (\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]\setminus\{i\}}$ be a tuple of random strings. For some subset $\mathcal{S} \subseteq [N]$, denote by $\mathbf{seed}_i[\mathcal{S}] = (\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in\mathcal{S}\setminus\{i\}}$. Also, let ZeroShare denote the algorithm (implicitly parameterized by PRF) that on input $\mathbf{seed}_i[\mathcal{S}]$ and arbitrary $\mathsf{str} \in \{0,1\}^{\ell+1}$ outputs

$$\mathsf{ZeroShare}(\mathbf{seed}_i[\mathcal{S}], \mathsf{str}) := \sum_{j\in\mathcal{S}\setminus\{i\}} (\mathsf{PRF}_{\mathsf{seed}_{j,i}}(\mathsf{str}) - \mathsf{PRF}_{\mathsf{seed}_{i,j}}(\mathsf{str})) \bmod p.$$

Observe that $\sum_{i\in\mathcal{S}} \mathsf{ZeroShare}(\mathbf{seed}_i[\mathcal{S}], \mathsf{str}) = 0$. As in [DKM+24], we employ ZeroShare to generate zero-shares in order to mask the user outputs in our distributed protocol. We describe our construction TCH in fig. 3.

Security Analysis. We start by giving some intuition for the security of our group-based construction.

Collision resistance. Observe that if it is possible to simulate the game without trapdoors $(\mathsf{sh}_i)_{i\in\mathcal{S}}$, then *any* collision allows to solve DL (either of H_0 or H_1). However, some private information seems required to simulate the DTrapColl oracle. This is because the image Y is determined *before* the input x^* in DTrapColl is provided. Therefore, the game must be able to produce a valid preimage of Y given any x^*. However, preimage resistance implies that this must be hard except if DTrapColl is invoked which relies on $(\mathsf{sh}_i)_{i\in\mathcal{S}}$.
The core insight is that it is sufficient to either know h_0^{-1} or h_1^{-1} to simulate DTrapColl. This is because we can open Y to x^* as follows

$$\begin{aligned} Y &= xG + r_0H_0 + r_1H_1 \\ &= x^*G + (h_0^{-1}\cdot(x-x^*) + r_0)H_0 + r_1H_1 \\ &= x^*G + r_0H_0 + (h_1^{-1}\cdot(x-x^*) + r_1)H_1. \end{aligned}$$

Also, the view of the adversary A is independent of whether h_0^{-1} or h_1^{-1} was employed for the simulation. Hence any collision-finder A allows to recover the DL of H_{1-b} if h_b^{-1} was used in the simulation. Interestingly, the commit-and-open protocol to establish $R = \sum_{i\in\mathcal{S}} R_i$ is not required for collision resistance.

Preimage resistance. Roughly, we must show that it is hard to find a preimage for Y output by AggEval without following the DTrapColl protocol. This seems to contradict the intuition given above: the simulator must be able to come up with arbitrary preimages for Y. However, the preimage found by A is for some Y established in an *unfinished* session. We can therefore guess which session is unfinished and embed a discrete logarithm challenge $\tilde{Y}$ into AggEval by employing the properties of COM. We show that simulation of unfinished sessions is possible even for such misformed outputs $\tilde{Y}$ in DEval. If we further know the discrete logarithm of H_0 and H_1, then any TCH opening to $\tilde{Y}$ allows to compute its discrete logarithm.

Image unbiasability. At a first glance, this seems to follows immediately by the properties of COM. While the corrupted users $k \in \mathcal{C}$ must commit to R_k

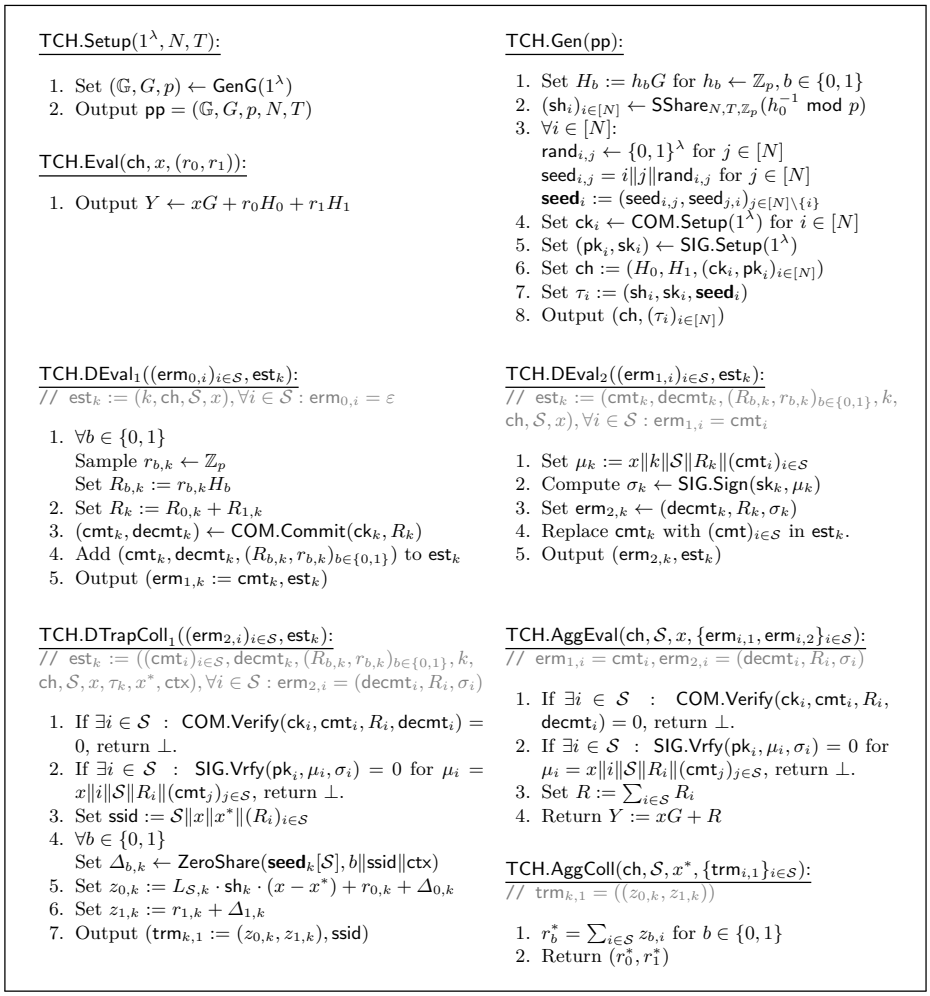

Fig. 3. Our group-based threshold chameleon hash. We assume that if parsing fails, then the algorithm outputs $\perp$. We denote by $L_{\mathcal{S},k}$ the Lagrange coefficient.

in the first round, honest user keys can be set up in equivocable mode, in which case the honest contributions R_k for $k \in \mathcal{H}$ are revealed only in the second round. As at least one R_k is honestly chosen in each signing session and has high min-entropy, the aggregated value $R = xG + \sum_{k\in\mathcal{S}} R_k$ for a priori chosen x will not appear twice, except with negligible probability.

However, this argument crucially relies on the views of all honest users in $\mathcal{S}$ being consistent in every session. We ensure that through attaching a signature on $x\|k\|\mathcal{S}\|R_k\|(\mathsf{cmt}_i)_{i\in\mathcal{S}}$ to the round message at the end of $\mathsf{TCH.DEval}_2$ and enforcing that $\mathsf{TCH.AggEval}$ and $\mathsf{TCH.DTrapColl}_1$ abort if one of the sig-

natures is not valid. This avoids correlations between sessions, and prevents that the adversary opens commitments freely in equivocation mode: Now, the adversary is forced to reuse the opening that was signed by the honest user. We now argue that any pair of sessions aggregates to the same value with at most negligible probability. There are two cases how an adversary could break image unbiasability:

Case 1: the same honest user aggregates to Y in different sessions. We guess this user k and set up the commitment key ck_k in equivocable mode, all other commitment keys we set up in extractable mode. The user k then samples commitments in $\mathsf{TCH.DEval}_1$ and equivocates them to a random nonce R_k in $\mathsf{TCH.DEval}_2$. Due to the signature check, the adversary must open the commitments $\mathsf{cmt}_k, \mathsf{cmt}_k$ of the two sessions via the $\mathsf{TCH.DEval}_2$ oracle and it obtains nonces R_k and R_k. As commitments of all the other users (including the honest users ones) are set up in extractable mode and input x is part of the users state, the respective nonces R_i are fixed already by the commitments input to the $\mathsf{TCH.DEval}_2$ oracle. Hence the choice of R_k determines the aggregated value Y. When R_k is later chosen at random in the $\mathsf{TCH.DEval}_2$ call of the other session, the probability that the aggregated value Y is equal to Y is negligible.

Case 2: different honest users aggregate to Y and Y in sessions with distinct user sets such that $Y = Y$. The same entropy argument as above works, however, we must identify appropriate calls to $\mathsf{TCH.DEval}_2$ carefully. Because the signer sets are different and bound to the commitments through the signature, there must be two distinct $\mathsf{TCH.DEval}_2$ calls that determine the aggregated Y and Y. As such, the nonces R_k and R_k in these distinct calls randomize Y and Y across sessions, dependent on the order of the $\mathsf{TCH.DEval}_2$ calls.

Note that the masking via ZeroShare implicitly ensures that users must agree on ssid (similar to [KRT24, LNÖ25]): if the values in ssid disagree, then the outputs by DTrapColl look uniform and therefore reveal no critical information.

Theorem 2. *The scheme in fig. 3 is a* $(2,1)-$*threshold chameleon hash satisfying correctness, collision resistance, preimage resistance and unbiasability under the security of* PRF*, extractability and equivocability of* COM*, unforgeability of* SIG*, and the* DL *assumption.*

Proof. Correctness is straightforward and follows from the definition of ZeroShare and SShare. That is, for $Y = xG + r_0H_0 + r_1H_1$ with $r_b = \sum_{k\in\mathcal{S}} r_{b,k}$, $b \in \{0,1\}$, we have

$$
\begin{aligned}
&x^*G + r_0^*H_0 + r_1^*H_1 \\
=&x^*G + (\sum_{k\in\mathcal{S}}(L_{\mathcal{S},k}\cdot \mathsf{sh}_k\cdot(x-x^*) + r_{0,k} + \Delta_{0,k}))H_0 + (\sum_{k\in\mathcal{S}}(r_{1,k}+\Delta_{1,k}))H_1 \\
=&x^*G + (h_0^{-1}\cdot(x-x^*) + \sum_{k\in\mathcal{S}} r_{0,k})H_0 + (\sum_{k\in\mathcal{S}} r_{1,k})H_1 \\
=&xG + r_0H_0 + r_1H_1 = Y
\end{aligned}
$$

as $\sum_{i\in\mathcal{S}} \Delta_{b,k} = 0$ and $h_0^{-1} = \sum_{k\in\mathcal{S}} L_{\mathcal{S},k} \cdot \mathsf{sh}_k$. In the remaining, let us prove security of TCH (cf. definition 6). For this, we define three events depending on the behavior of adversary A.

- $\mathsf{E}_{\mathsf{A},1}$: the event that A breaks collision resistance, *i.e.*, $\mathsf{Eval}(\mathsf{ch}, x, (r_0, r_1)) = \mathsf{Eval}(\mathsf{ch}, x^*, (r_0^*, r_1^*))$ and $(x, (r_0, r_1)) \neq (x^*, (r_0^*, r_1^*))$.
- $\mathsf{E}_{\mathsf{A},2}$: the event that A breaks preimage resistance, *i.e.*, $(y, \cdot, \cdot) \in \mathsf{Agg}$ and $y \notin \mathsf{Cmpl}$ for $y = \mathsf{Eval}(\mathsf{ch}, x^*, (r_0^*, r_1^*))$.
- $\mathsf{E}_{\mathsf{A},3}$: the event that A breaks image unbiasability, *i.e.*, $\mathsf{dCollision} = \mathsf{true}$.

Below, we assume that one of the above events occurs and bound the probability of this event occurring with negligible probability. The statement follows.

Below, let $N > T \in \mathbb{N}$ and let A be a PPT adversary against security of TCH. We denote by $\mathcal{C}$ the set of corrupted users, and by $\mathcal{H}$ the set of honest users. For any $\mathcal{S} \subseteq [N]$, denote $\mathcal{S}_\mathcal{C} = \mathcal{S} \cap \mathcal{C}$ and $\mathcal{S}_\mathcal{H} = \mathcal{S} \cap \mathcal{H}$. Also, we denote by Q the number of started sessions. We proceed via game hops and denote by ε_i the probability that A wins in Game i. For $\ell \in \{1, 2, 3\}$, we denote by $\varepsilon_{\ell.0}$ the probability that A triggers event $\mathsf{E}_{\mathsf{A},\ell}$. Thus, the probability that A breaks security of the construction can be bounded by $\varepsilon := \sum_{\ell\in\{1,2,3\}} \varepsilon_{\ell.0}$. We proceed via game hops to bound the probabilities $\varepsilon_{\ell.0}$, where we define starting games Game $\ell.0$ for each ℓ as the honest game $\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}(1^\lambda, N, T)$ and denote by $\varepsilon_{\ell.i}$ the probability that A triggers event $\mathsf{E}_{\mathsf{A},\ell}$ in Game $\ell.i$.

Collision resistance. Let us show collision resistance. Below, we bound the probability that event $\mathsf{E}_{\mathsf{A},1}$ occurs.

Game 1.0. This is the honest game $\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}(1^\lambda, N, T)$. By definition, the adversary A succeeds in Game 1.0 with probability at most

$$\varepsilon_{1.0} := \Pr[\mathsf{E}_{\mathsf{A},1}].$$

Game 1.1. In this game, the game aborts if the adversary's collision $(x, (r_0, r_1), x^*, (r_0^*, r_1^*))$ is valid and satisfies $r_1 - r_1^* \neq 0$. Call this event E_1.
Let us show that $\Pr[E_1] = \mathsf{negl}(\lambda)$ under the DL assumption. For this, observe that h_1 is not needed to simulate Game 1.0 except to setup H_1. Therefore, the DL reduction simply sets H_1 equal to its DL challenge. If $r_1 - r_1^* \neq 0$ in A's collision, then the reduction outputs $(x - x^* + (r_0 - r_0^*)h_0) \cdot (r_1^* - r_1)^{-1}$ as the DL for H_1. This will be a valid DL for H_1 as we have

$$\begin{aligned} & xG + r_0H_0 + r_1H_1 = x^*G + r_0^*H_0 + r_1^*H_1 \\ \implies & (x - x^*)G + (r_0 - r_0^*)h_0G = (r_1^* - r_1)H_1 \\ \implies & (x - x^* + (r_0 - r_0^*)h_0) \cdot (r_1^* - r_1)^{-1}G = H_1, \end{aligned}$$

Moreover, we know that $(x - x^* + (r_0 - r_0^*)h_0) \neq 0$. This follows because if $(x - x^* + (r_0 - r_0^*)h_0) = 0$ then $(r_1^* - r_1)$ would be a zero divisor, which cannot be possible because $(r_1^* - r_1) \in \mathbb{Z}_p \setminus \{0\}$ and p prime. Hence the

reduction outputs a valid DL for H_1 and hence the reduction's advantage on DL is identical to $\Pr[E_1]$, i.e.,

$$|\varepsilon_{1.0} - \varepsilon_{1.1}| \leq \mathsf{negl}(\lambda).$$

Next, our goal is to show that event E_0, defined as $r_0 - r_0^* \neq 0$, also does not occur except with negligible probability. If so, then A's advantage on finding a collision is 0 as E_0 and E_1 does not occur. For this, our goal is to simulate the game via h_1 without knowing h_0.

Note that ssid is unique in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ for user k (*i.e.*, user k never obtains the same ssid twice in a non-aborting call to $\mathcal{O}^{\mathsf{DTrapColl}_1}$), except if the game samples twice an identical R_k in $\mathcal{O}^{\mathsf{DEval}_1}$. The latter occurs with probability at most Q^2/p, where Q is the number of $\mathcal{O}^{\mathsf{DEval}_1}$ invocations, by a standard birthday bound argument. In consequence, every honest user executes $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid at most once. In the following, we use this fact implicitly to define values for each user with respect to ssid that occurs in $\mathcal{O}^{\mathsf{DTrapColl}_1}$. By the above discussion, these values are well-defined.

Game 1.2. The game initializes empty tables $\mathsf{UnCollHS}[\cdot] = \bot, \mathsf{PartialMask}[\cdot] = \bot, \mathsf{Mask}[\cdot] = \bot$ and $\mathsf{MaskedResp}[\cdot] = \bot$ in the beginning. These tables are updated in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ to store the following.

- $\mathsf{UnCollHS}[\mathsf{ssid}, \mathsf{ctx}]$ stores the set of honest users that have not executed $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid yet.
- $\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, i, j]$ stores the partial masks $m_{b,i,j} = \mathsf{PRF}_{\mathsf{seed}_{i,j}}(b \| \mathsf{ssid} \| \mathsf{ctx})$ for honest users $i, j \in \mathcal{H}$ used within $\mathsf{ZeroShare}$.
- $\mathsf{Mask}[b, \mathsf{ssid}, \mathsf{ctx}, k]$ stores the zero share $\Delta_{b,k}$.
- $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k]$ stores the masked response $z_{b,k}$.

Recall that $\mathcal{S}_\mathcal{C} = \mathcal{S} \cap \mathcal{C}$ and $\mathcal{S}_\mathcal{H} = \mathcal{S} \cap \mathcal{H}$. We also unroll the definition of $\mathsf{ZeroShare}$, *i.e.*, the game samples

$$\Delta_{b,k} \leftarrow \sum_{j \in \mathcal{S}_\mathcal{C}} (\mathsf{PRF}_{\mathsf{seed}_{k,j}}(b \| \mathsf{ssid} \| \mathsf{ctx}) - \mathsf{PRF}_{\mathsf{seed}_{j,k}}(b \| \mathsf{ssid} \| \mathsf{ctx})) + \sum_{j \in \mathcal{S}_\mathcal{H}} (\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, k, j] - \mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, j, k]),$$

and stores $\Delta_{b,k}$ in $\mathsf{Mask}[b, \mathsf{ssid}, \mathsf{ctx}, k]$. This game change is purely syntactic if ssid is unique (as discussed above), and we have

$$|\varepsilon_{1.1} - \varepsilon_{1.2}| \leq Q^2/p.$$

Game 1.3. The game samples random values for the partial masks $m_{b,i,j}$ stored in $\mathsf{PartialMask}$ for honest users $i, j \in \mathcal{H}$ and $b \in \{0, 1\}$. That is, if $\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, i, j] = \bot$, it samples $m_{b,i,j} \leftarrow \mathbb{Z}_p$ and sets $\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, i, j] = m_{b,i,j}$.

Clearly, we have by security of PRF that

$$|\varepsilon_{1.2} - \varepsilon_{1.3}| \leq \mathsf{negl}(\lambda).$$

Game 1.4. The game samples $\Delta_{b,k}$ directly either at random or consistently for the last honest user in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid and ctx. That is, it sets $\widetilde{\mathcal{S}_\mathcal{H}} \leftarrow \mathsf{UnCollHS}[\mathsf{ssid}, \mathsf{ctx}]$ and if $\widetilde{\mathcal{S}_\mathcal{H}} \neq \{k\}$, it samples $\Delta_{b,k} \leftarrow \mathbb{Z}_p$ and sets $\mathsf{Mask}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow \Delta_{b,k}$. Else, k is the last honest user by definition of $\mathsf{UnCollHS}$, and the game samples the mask consistently. In particular, it samples

$$\mathsf{Mask}[b, \mathsf{ssid}, \mathsf{ctx}, k] = -\sum_{j \in \mathcal{S}_\mathcal{H} \setminus \{k\}} \mathsf{Mask}[b, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{C}} \Delta_{b,j},$$

where $\Delta_{b,j} = \mathsf{ZeroShare}(\mathbf{seed}_j[\mathcal{S}], b \| \mathsf{ssid} \| \mathsf{ctx})$ for $j \in \mathcal{C}$.
Let us show that Game 1.3 and Game 1.4 are identically distributed. Observe that in Game 1.3, the value $\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, k, j]$ and $\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, j, k]$ is distributed uniformly at random if there is some $j \in \widetilde{\mathcal{S}_\mathcal{H}} \setminus \{k\}$, as in this case these entries were not initialized yet, and therefore $m_{b,k,j}$ and $m_{b,j,k}$ are freshly sampled at random. On the other hand, if $\widetilde{\mathcal{S}_\mathcal{H}} = \{k\}$, then k is the last honest user with respect to ssid and $\mathsf{PartialMask}[b, \mathsf{ssid}, \mathsf{ctx}, i, j]$ is defined for all $j, i \in \mathcal{S}_\mathcal{H}$. As it holds that $\sum_{j \in \mathcal{S}} \Delta_{b,j} = 0$, where $\Delta_{b,j} = \mathsf{Mask}[b, \mathsf{ssid}, \mathsf{ctx}, j]$ for $j \in \mathcal{S}_\mathcal{H}$ and $\Delta_{b,j} = \mathsf{ZeroShare}(\mathbf{seed}_j[\mathcal{S}], b \| \mathsf{ssid} \| \mathsf{ctx})$ for $j \in \mathcal{S}_\mathcal{C}$, it must hold that

$$\Delta_{b,k} = -\sum_{j \in \mathcal{S} \setminus \{k\}} \Delta_{b,j}. \tag{2}$$

This is exactly the distribution of $\Delta_{b,k}$ in Game 1.4, and we have

$$\varepsilon_{1.3} = \varepsilon_{1.4}.$$

Note that at this point, the values stored in $\mathsf{PartialMask}[\cdot]$ are not referenced anymore. Therefore, we do not ensure that $\mathsf{PartialMask}$ is setup consistently in the following games.

Game 1.5. The game samples the response $z_{b,k}$ either at random or consistently for the last user in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid. In more detail, denote $\mathsf{sh}_0^* = h_0^{-1} \bmod p$. To simulate $\mathcal{O}^{\mathsf{DTrapColl}_1}$ for honest user k, the game sets $z_{b,k} \leftarrow \mathbb{Z}_p$ and $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow z_{b,k}$ if $\widetilde{\mathcal{S}_\mathcal{H}} \neq \{k\}$, where $\widetilde{\mathcal{S}_\mathcal{H}} \leftarrow \mathsf{UnCollHS}[\mathsf{ssid}, \mathsf{ctx}]$, and else it sets $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow z_{b,k}$ for

$$\begin{aligned}
z_{0,k} =& (x - x^*)\mathsf{sh}_0^* - (x - x^*) \sum_{j \in \mathcal{S}_\mathcal{C}} L_{\mathcal{S},j} \mathsf{sh}_j + \sum_{j \in \mathcal{S}_\mathcal{H}} r_{0,j} \\
&- \sum_{j \in \mathcal{S}_\mathcal{H} \setminus \{k\}} \mathsf{MaskedResp}[0, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{S}_\mathcal{C}} \Delta_{0,j}, \\
z_{1,k} =& \sum_{j \in \mathcal{S}_\mathcal{H}} r_{1,j} - \sum_{j \in \mathcal{S}_\mathcal{H} \setminus \{k\}} \mathsf{MaskedResp}[1, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{S}_\mathcal{C}} \Delta_{1,j},
\end{aligned}$$

where $\Delta_{b,j} = \mathsf{ZeroShare}(\mathbf{seed}_j[\mathcal{S}], b \| \mathsf{ssid} \| \mathsf{ctx})$ for $j \in \mathcal{C}$ and $r_{b,j}$ are the discrete logarithms of $R_{b,j}$ for honest users $j \in \mathcal{S}_\mathcal{H}$.

Let us show that Game 1.4 and Game 1.5 are identically distributed. Let us consider an intermediate game between Game 1.4 and Game 1.5, where the game samples

$$\begin{aligned}\Delta_{0,k} &:= \Delta^*_{0,k} - (L_{\mathcal{S},k} \cdot \mathsf{sh}_k \cdot (x - x^*) + r_{0,k})\\ \Delta_{1,k} &:= \Delta^*_{1,k} - r_{1,k}\end{aligned}$$

for $\Delta^*_{b,k} \leftarrow \mathbb{Z}_p$ if $\widetilde{\mathcal{S}_{\mathcal{H}}} \neq \{k\}$, else it sets $\Delta_{b,k}$ as in eq. (2). Clearly, this intermediate game is distributed as in Game 1.4.
On the other hand, a simple calculation yields that $z_{0,k} = L_{\mathcal{S},k} \cdot \mathsf{sh}_k \cdot (x - x^*) + r_{0,k} + \Delta_{0,k} = \Delta^*_{0,k}$ and $z_{1,k} = \Delta^*_{1,k}$ for $\widetilde{\mathcal{S}_{\mathcal{H}}} \neq \{k\}$, which is the same distribution as in Game 1.5. Also, if $\widetilde{\mathcal{S}_{\mathcal{H}}} = \{k\}$, then we have in the intermediate game that

$$\begin{aligned}z_{0,k} &= L_{\mathcal{S},k} \cdot \mathsf{sh}_k \cdot (x - x^*) + r_{0,k} + \Delta_{0,k}\\ &= L_{\mathcal{S},k} \cdot \mathsf{sh}_k \cdot (x - x^*) + r_{0,k} - \sum_{j \in \mathcal{S} \setminus \{k\}} \Delta_{0,j}\\ &= L_{\mathcal{S},k} \cdot \mathsf{sh}_k \cdot (x - x^*) + r_{0,k} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \Delta_{0,j} - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{0,j}\\ &= L_{\mathcal{S},k} \cdot \mathsf{sh}_k \cdot (x - x^*) + r_{0,k} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \left(\Delta^*_{0,j} - (L_{\mathcal{S},j} \cdot \mathsf{sh}_j \cdot (x - x^*) + r_{0,j})\right) - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{0,j}\\ &= (x - x^*)(\mathsf{sh}^*_0 - \sum_{j \in \mathcal{S}_{\mathcal{C}}} L_{\mathcal{S},j} \mathsf{sh}_j) + \sum_{j \in \mathcal{S}_{\mathcal{H}}} r_{0,j} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \Delta^*_{0,j} - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{0,j}\end{aligned}$$

In the second to last equality, we used the fact that x^* is fixed in ssid, in particular, it coincides for each honest user for $\mathcal{O}^{\mathsf{DTrapColl}_1}$ calls with ssid, and that $\sum_{j \in \mathcal{S}} L_{\mathcal{S},j} \mathsf{sh}_j = \mathsf{sh}^*_0$. As above, we also obtain

$$\begin{aligned}z_{1,k} &= r_{1,k} + \Delta_{1,k}\\ &= r_{1,k} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \Delta_{1,j} - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{1,j}\\ &= r_{1,k} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \left(\Delta^*_{1,j} - r_{1,j}\right) - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{1,j}\\ &= \sum_{j \in \mathcal{S}_{\mathcal{H}}} r_{i,j} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \Delta^*_{1,j} - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{1,j}\end{aligned}$$

It follows that the intermediate game is also distributed identically to Game 1.5, and we have

$$\varepsilon_{1.4} = \varepsilon_{1.5}.$$

Game 1.6. The game samples the consistent response $z_{b,k}$ with respect to ssid and ctx for the last honest user k in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ differently. In more detail,

denote $\mathsf{sh}_1^* = h_1^{-1} \bmod p$. As before, in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ for honest user $k \in \mathcal{S}_\mathcal{H}$, the game sets $z_{b,k} \leftarrow \mathbb{Z}_p$ and $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow z_{b,k}$ if $\widetilde{\mathcal{S}_\mathcal{H}} \neq \{k\}$, where $\widetilde{\mathcal{S}_\mathcal{H}} \leftarrow \mathsf{UnCollHS}[\mathsf{ssid}, \mathsf{ctx}]$. But if $\widetilde{\mathcal{S}_\mathcal{H}} = \{k\}$, *i.e.*, k is the last user to answer $\mathcal{O}^{\mathsf{DTrapColl}_1}$ for ssid, then the game sets $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow z_{b,k}$ for

$$z_{0,k} = -(x - x^*) \sum_{j \in \mathcal{S}_\mathcal{C}} L_{\mathcal{S},j} \mathsf{sh}_j + \sum_{j \in \mathcal{S}_\mathcal{H}} r_{0,j} - \sum_{j \in \mathcal{S}_\mathcal{H} \setminus \{k\}} \mathsf{MaskedResp}[0, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{S}_\mathcal{C}} \Delta_{0,j},$$
$$z_{1,k} = \mathsf{sh}_1^* \cdot (x - x^*) + \sum_{j \in \mathcal{S}_\mathcal{H}} r_{1,j} - \sum_{j \in \mathcal{S}_\mathcal{H} \setminus \{k\}} \mathsf{MaskedResp}[1, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{S}_\mathcal{C}} \Delta_{1,j}.$$

Let us inspect the distribution of $z_{b,k}$ in both games if $\widetilde{\mathcal{S}_\mathcal{H}} = \{k\}$, otherwise the distribution remains identical by construction. We can write

$$\begin{aligned} R_k &= r_{0,k} H_0 + r_{1,k} H_1 \\ &= r_{0,k} H_0 - (x - x^*) G + (x - x^*) G + r_{1,k} H_1 \\ &= \underbrace{(r_{0,k} - \mathsf{sh}_0^* \cdot (x - x^*))}_{r_{0,k}^*} H_0 + \underbrace{(r_{1,k} + \mathsf{sh}_1^* \cdot (x - x^*))}_{r_{1,k}^*} H_1. \end{aligned}$$

Observe that $r_{0,k}^*$ and $r_{1,k}^*$ are distributed as $r_{0,k}$ and $r_{1,k}$ in Game 1.5. Furthermore, using $r_{0,k}^*$ and $r_{1,k}^*$ instead of $r_{0,k}$ and $r_{1,k}$ in the definition of $z_{b,k}$ in Game 1.5, we obtain the distribution in Game 1.6. The distribution of R_k remains unchanged. Therefore, we have

$$\varepsilon_{1.5} = \varepsilon_{1.6}.$$

Game 1.7. In this game, the shares sh_i of corrupt users $i \in \mathcal{C}$ are sampled uniformly at random.
Observe that in Game 1.6, the game is simulated without the shares sh_i of honest users $i \in \mathcal{H}$. Due to the properties of Shamir's secret sharing, each share sh_i for $i \in \mathcal{C}$ is distributed at random, as in Game 1.7. Therefore, we have

$$\varepsilon_{1.6} = \varepsilon_{1.7}.$$

Finally, observe that simulation in Game 1.7 does not rely on sh_0^*. Therefore, we can argue as in Game 1.1 that $\Pr[E_0] = \mathsf{negl}(\lambda)$ under the DL assumption. Collecting the above bounds, we obtain

$$\Pr[\mathsf{E}_{\mathsf{A},1}] = \mathsf{negl}(\lambda).$$

Preimage resistance. We bound the probability that event $\mathsf{E}_{\mathsf{A},2}$ occurs.

Game 2.0. This is the honest game $\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}(1^\lambda, N, T)$. By definition, the adversary A succeeds in Game 2.0 with probability

$$\varepsilon_{2.0} := \Pr[\mathsf{E}_{\mathsf{A},2}].$$

Game 2.1. In the beginning, the game initializes empty tables $\mathsf{UnCollHS}[\cdot] = \bot$ and $\mathsf{MaskedResp}[\cdot] = \bot$. As in Game 1.2, These tables are updated in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ to store the following.

- $\mathsf{UnCollHS}[\mathsf{ssid}, \mathsf{ctx}]$ stores the set of honest users that have not executed $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid yet.
- $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k]$ stores the masked response $z_{b,k}$.

Further, the game samples the masked response $z_{b,k}$ either at random or consistently for the last user in $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid and ctx as in Game 1.5. That is, denote $\mathsf{sh}_0^* = h_0^{-1} \bmod p$. To simulate $\mathcal{O}^{\mathsf{DTrapColl}_1}$ for honest user k, the game sets $z_{b,k} \leftarrow \mathbb{Z}_p$ and $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow z_{b,k}$ if $\widetilde{\mathcal{S}_{\mathcal{H}}} \neq \{k\}$, where $\widetilde{\mathcal{S}_{\mathcal{H}}} \leftarrow \mathsf{UnCollHS}[\mathsf{ssid}, \mathsf{ctx}]$, and else it sets $\mathsf{MaskedResp}[b, \mathsf{ssid}, \mathsf{ctx}, k] \leftarrow z_{b,k}$ for

$$\begin{aligned} z_{0,k} =& (x - x^*)\mathsf{sh}_0^* - (x - x^*) \sum_{j \in \mathcal{S}_{\mathcal{C}}} L_{\mathcal{S},j}\mathsf{sh}_j + \sum_{j \in \mathcal{S}_{\mathcal{H}}} r_{0,j} \\ & - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \mathsf{MaskedResp}[0, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{0,j}, \\ z_{1,k} =& \sum_{j \in \mathcal{S}_{\mathcal{H}}} r_{1,j} - \sum_{j \in \mathcal{S}_{\mathcal{H}} \setminus \{k\}} \mathsf{MaskedResp}[1, \mathsf{ssid}, \mathsf{ctx}, j] - \sum_{j \in \mathcal{S}_{\mathcal{C}}} \Delta_{1,j}, \end{aligned}$$

where $\Delta_{b,j} = \mathsf{ZeroShare}(\mathbf{seed}_j[\mathcal{S}], b \| \mathsf{ssid} \| \mathsf{ctx})$ for $j \in \mathcal{C}$ and $r_{b,j}$ are the discrete logarithms of $R_{b,j}$ for honest users $j \in \mathcal{S}_{\mathcal{H}}$.
As in the proof of collision resistance in Game 1.2 to Game 1.5, we can show that

$$|\varepsilon_{2.0} - \varepsilon_{2.1}| \leq \mathsf{negl}(\lambda).$$

Observe that as consequence, the discrete logarithms of $R_{0,k}$ and $R_{1,k}$ for $k \in \mathcal{S}_{\mathcal{H}}$ are only required for simulation when the last user answers $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with consistent ssid and ctx. Further, recall that $\mathsf{ssid} = \mathcal{S} \| x \| x^* \| (R_i)_{i \in \mathcal{S}}$. Observe that $Y = xG + \sum_{i \in \mathcal{S}} R_i$ determines Y and note that Y must have been aggregated to in $\mathcal{O}^{\mathsf{AggEval}}$. Furthermore, the fact that $\mathsf{E}_{\mathsf{A},2}$ occurs implies two important events:

1. It holds that $(Y, \cdot, \cdot) \in \mathsf{Agg}$ and $Y \notin \mathsf{Cmpl}$ for $Y = \mathsf{Eval}(\mathsf{ch}, x^*, (r_0^*, r_1^*))$. By definition, there is some honest user $k_{\mathsf{dcoll}} \in \mathcal{S}_{\mathcal{H}}$ such that $k_{\mathsf{dcoll}} \notin \mathsf{SSID}[\mathsf{ssid}, \mathsf{ctx}, \mathcal{S}_{\mathcal{H}}, Y]$, that is, honest user k_{dcoll} did not yet execute $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with $(\mathsf{ssid}, \mathsf{ctx})$, where $\mathcal{S}_{\mathcal{H}} \in \mathcal{S}$ and Y is determined by ssid.
2. As $(Y, \cdot, \cdot) \in \mathsf{Agg}$, there must be a session for which $\mathcal{O}^{\mathsf{AggEval}}$ was invoked for some honest user $k_{\mathsf{deval}} \in \mathcal{S}_{\mathcal{H}}$ that aggregated to Y, *i.e.*, $\mathcal{O}^{\mathsf{DEval}_2}$ was executed for k_{deval} and the other contributions $\mathsf{erm}_{i,2} = R_i$ for $i \in \mathcal{S} \setminus k_{\mathsf{deval}}$ are determined and yield $Y = xG + \sum_{i \in \mathcal{S}} R_i$. Also, the commitments must have been opened to values extracted in $\mathcal{O}^{\mathsf{DEval}_2}$ when invoked for user k_{deval}.

Next, we will guess this unfinished session and embed a DL challenge D.

Game 2.2. The game initially guesses a session $\mathsf{sid} \in [Q]$ and the honest user $k_{\mathsf{deval}} \in \mathcal{H}$ as described above. It aborts in the end if user k_{deval} did not aggregate to Y in session sid, where $Y = \mathsf{Eval}(\mathsf{ch}, x^*, (r_0^*, r_1^*))$. Also, if user k_{deval} executes $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid and ctx in session sid, then the game aborts if all honest users $i \in \mathcal{S}_{\mathcal{H}}$ executed $\mathcal{O}^{\mathsf{DTrapColl}_1}$ with ssid and ctx.
The game aborts if the guess for k_{deval} or sid is incorrect. As such a user and session must exist by the above discussion, the guess is correct with probability $1/(T \cdot Q)$. Therefore we have

$$\varepsilon_{2.1} \leq T \cdot Q \cdot \varepsilon_{2.2}$$

Game 2.3. This game sets up the commitment keys ck_i differently. That is, upon receiving the list of corrupted users $\mathcal{C}$, the game generates all commitment keys ck_i of COM for $i \in [N] \setminus \{k_{\mathsf{deval}}\}$ with $\mathsf{COM.ExtSetup}$.
By a standard hybrid argument and extractabilityof COM, it holds

$$|\varepsilon_{2.2} - \varepsilon_{2.3}| \leq \mathsf{negl}(\lambda).$$

Game 2.4. Now, the game extracts R_i' from the commitments cmt_i for all parties $i \in \mathcal{S} \setminus \{k_{\mathsf{deval}}\}$ in $\mathcal{O}^{\mathsf{DEval}_2}$ for session sid and user $k = k_{\mathsf{deval}}$. Further, the $\mathcal{O}^{\mathsf{AggEval}}$ oracle aborts if there exists an $i \in \mathcal{S} \setminus \{k_{\mathsf{deval}}\}$ such that $R_i \neq R_i'$, where R_i' was extracted in $\mathcal{O}^{\mathsf{DEval}_2}$. The extractability property of COM yields that the abort probability of $\mathcal{O}^{\mathsf{AggEval}}$ is negligible, therefore we have

$$|\varepsilon_{2.3} - \varepsilon_{2.4}| \leq \mathsf{negl}(\lambda).$$

Game 2.5. The game initially sets up $\mathsf{ck}_{k_{\mathsf{deval}}}$ via $(\mathsf{ck}_{k_{\mathsf{deval}}}, \mathsf{td}) \leftarrow \mathsf{COM.EqvSetup}$. In $\mathcal{O}^{\mathsf{DEval}_1}$, the game sets $(\mathsf{cmt}_k, \mathsf{eqtd}_k) \leftarrow \mathsf{SimCom}(\mathsf{ck}_k, \mathsf{td})$ instead of committing to R_k for $k = k_{\mathsf{deval}}$ in session sid. In $\mathcal{O}^{\mathsf{DEval}_2}$ in session sid, the game samples $D \leftarrow \mathbb{G}$ and sets $R_k := D - xG - \sum_{i \in \mathcal{S} \setminus \{k\}} R_i$ for $k = k_{\mathsf{deval}}$, where R_i is extracted from cmt_i. Then, it equivocates the commitment cmt_k to obtain an opening $\mathsf{decmt}_k \leftarrow \mathsf{Equivocate}(\mathsf{td}_k, \mathsf{cmt}_k, \mathsf{eqtd}_k, R_k)$ to R_k for cmt_k. Otherwise, the game proceeds as in Game 2.4.
Observe that R_k is distributed as in Game 2.4. Also, as discussed above, the discrete logarithm of R_k is not required for simulation. Thus, we have under equivocability of COM that

$$|\varepsilon_{2.2} - \varepsilon_{2.3}| \leq \mathsf{negl}(\lambda).$$

Finally, let us show that a successful adversary A in Game 2.5 allows us to construct an adversary B on the DL assumption. In particular, B simulates Game 2.5 to A and embeds its DL challenge D as described in Game 2.5 in session sid. When A outputs its solution, B outputs $x^* + r_0^* \cdot h_0 + r_1^* \cdot h_1$ to the DL challenger.

As $\mathsf{E}_{\mathsf{A},2}$ occurs, we have that

$$Y = \mathsf{Eval}(\mathsf{ch}, x^*, (r_0^*, r_1^*)) = (x^* + r_0^* \cdot h_0 + r_1^* \cdot h_1)G.$$

Further, we know that $Y = xG + \sum_{i \in \mathcal{S}} R_i$ as computed in $\mathcal{O}^{\mathsf{AggEval}}$. By definition of $R_{k_{\mathsf{deval}}}$, it holds that $Y = D$. This concludes the proof of preimage resistance.

Image unbiasability. We bound the probability that event $\mathsf{E}_{\mathsf{A},3}$ occurs and provide a negligible upper bound. That is, a user k aggregated twice to Y in distinct sessions or Y was aggregated to by distinct honest user sets $\mathcal{S}_{\mathcal{H}} \subseteq \mathcal{S}$ and $\mathcal{S}'_{\mathcal{H}} \subseteq \mathcal{S}'$.

Game 3.0. This is the honest game $\mathsf{Game}^{\mathsf{ch-sec}}_{\mathsf{TCH},\mathsf{A}}(1^\lambda, N, T)$. By definition, the adversary A succeeds in Game 3.0 with probability at least

$$\varepsilon_{3.0} := \Pr[\mathsf{E}_{\mathsf{A},3}].$$

Game 3.1. This game sets up all commitment keys in extractable mode. That is, the game initially samples all ck_i for $i \in [N]$ via $\mathsf{ExtSetup}$. By the extractability property of COM, it holds

$$|\varepsilon_{3.0} - \varepsilon_{3.1}| \leq \mathsf{negl}(\lambda).$$

Game 3.2. This game aborts if there exists an honest user $i \in \mathcal{H}$ who produces the same commitment cmt_i in two different sessions. To see that this game and the previous one are indistinguishable, note that since values $r_{b,i} \leftarrow \mathbb{Z}_p$ are sampled freshly and uniformly at random in each session, with all but negligible probability, user i is committing to a different value R_i in each session. Extractability of the commitment scheme then guarantees that with high probability an honest user does not sample the same commitment twice. Hence

$$|\varepsilon_{3.1} - \varepsilon_{3.2}| \leq \mathsf{negl}(\lambda).$$

Game 3.3 In $\mathcal{O}^{\mathsf{AggEval}}$, the game aborts if there exists an honest user $i \in \mathcal{S}_{\mathcal{H}}$ such that σ_i is a valid signature for $\mu_i = x \| \mathcal{S} \| R_i \| (\mathsf{cmt}_j)_{j \in \mathcal{S}}$, but μ_i was never signed before in $\mathcal{O}^{\mathsf{DEval}_2}$ by user i. We have under EUF-CMA security of SIG that

$$|\varepsilon_{3.2} - \varepsilon_{3.3}| \leq \mathsf{negl}(\lambda).$$

Game 3.4. Now, the oracle $\mathcal{O}^{\mathsf{DEval}_2}$ extracts R'_i from the commitments cmt_i for all corrupt parties $i \in \mathcal{S}_{\mathcal{C}}$. The game aborts if in the $\mathcal{O}^{\mathsf{AggEval}}$ oracle there exists a malicious user $i \in \mathcal{S}_{\mathcal{C}}$ such that cmt_i, R_i passes verification, but $R_i \neq R'_i$. In other words, if a malicious user opens cmt_i to a different value than to the extracted one.
By the extractability property of COM, this occurs with at most negligible probability.

$$|\varepsilon_{3.3} - \varepsilon_{3.4}| \leq \mathsf{negl}(\lambda).$$

Let us make couple of remarks about the implications of the introduced aborts.

Remark 5. Assume that $\mathcal{O}^{\mathsf{AggEval}}$ is queried for two sessions sid and sid' with distinct user sets $\mathcal{S} \neq \mathcal{S}'$, and neither call to $\mathcal{O}^{\mathsf{AggEval}}$ aborts. Then for each honest user in $i \in \mathcal{S}_{\mathcal{H}} \cap \mathcal{S}'_{\mathcal{H}}$ the corresponding commitments cmt_i and cmt'_i in the two $\mathcal{O}^{\mathsf{AggEval}}$ queries must be distinct.

This follows from the aborts introduced in Games 3.2 and 3.3: There are no repeating commitments sampled by honest users and any non-aborting $\mathcal{O}^{\mathsf{AggEval}}$ query cannot contain a SIG-forgery.

Remark 6. Assume that $\mathcal{O}^{\mathsf{AggEval}}$ is queried for session sid and honest party k (which defines x and $\mathcal{S}$) and round messages specifying $(\mathsf{cmt}_i, \mathsf{decmt}_i, R_i, \sigma_i)_{i\in\mathcal{S}}$, and the $\mathcal{O}^{\mathsf{AggEval}}$ call does not abort. Then all honest users $k \in \mathcal{S}_{\mathcal{H}}$ have opened their commitment cmt_k to R_k in a call to $\mathcal{O}^{\mathsf{DEval}_2}$ and the user set, input and commitments are equal to $\mathcal{S}$, x, $(\mathsf{cmt}_i)_{i\in\mathcal{S}}$ in the $\mathcal{O}^{\mathsf{DEval}_2}$ call.

This holds because all σ_i verify if $\mathcal{O}^{\mathsf{AggEval}}$ does not abort. Therefore, message $\mu_i = x\|i\|\mathcal{S}\|R_i\|(\mathsf{cmt}_j)_{j\in\mathcal{S}}$ was signed before by user i in $\mathcal{O}^{\mathsf{DEval}_2}$ (due to the abort condition added in game 3.3).

Now, let us argue that adversary A triggers event $\mathsf{E}_{\mathsf{A},3}$ in Game 3.4 with at most negligible probability. If event $\mathsf{E}_{\mathsf{A},3}$ occurs then, it holds that $\mathsf{dCollision} = \mathsf{true}$. Thus, there must exist an honest user k and a session sid such that when user k aggregates in session sid with user set $\mathcal{S}$ to $Y = xG + \sum_{j\in\mathcal{S}} R_j$ one of the following cases must hold:

Case 1: $(Y, \cdot, k) \in \mathsf{Agg}$, *i.e.*, user k aggregated to Y also in a second session $\mathsf{sid}' \neq \mathsf{sid}$;

Case 2: there exists a user set $\mathcal{S}'$ such that $k \notin \mathcal{S}'$ and $(Y, \mathcal{S}', \cdot) \in \mathsf{Agg}$, *i.e.*, some honest user $k' \in \mathcal{S}'$ aggregated to Y with different user set $\mathcal{S}' \neq \mathcal{S}$.

Case 1. Let us argue that Case 1 does not occur in Game 3.4 except with negligible probability. Before doing so, we introduce additional games where we guess the user k (as described above) and set up its commitment key in equivocal mode. Looking ahead, we will then leverage the entropy in the committed nonces R_k sampled by k to argue that event $\mathsf{E}_{\mathsf{A},3}$ occurs with negligible probability. Below, denote by $\varepsilon_{3.4.1}$ the probability that Case 1 occurs in Game 3.4. For $i > 4$, we denote by $\varepsilon_{3.i.1}$ the probability that Case 1 occurs in Game 3.i.1.

Game 3.5.1. Upon receiving the list of corrupted users $\mathcal{C}$ the game samples $k_{\$} \leftarrow \mathcal{H}$. When Case 1 occurs for the first time (if it occurs at all), the game aborts if the party k triggering this case is not $k_{\$}$, i.e., $k \neq k_{\$}$. As there are at most N honest users, we have that

$$\varepsilon_{3.4.1} = |\mathcal{H}| \cdot \varepsilon_{3.5.1} \leq N \cdot \varepsilon_{3.5.1}.$$

Game 3.6.1. The game sets up the commitment key of user $k_{\$}$ in equivocation mode, *i.e.*, it samples $\mathsf{ck}_{k_{\$}}$ using $\mathsf{EqvSetup}$. By the equivocation property of COM, it holds

$$|\varepsilon_{3.5.1} - \varepsilon_{3.6.1}| \leq \mathsf{negl}(\lambda).$$

Game 3.7.1 When $\mathcal{O}^{\mathsf{DEval}_1}$ is invoked for user $k = k_{\$}$, the game sets $(\mathsf{cmt}_k, \mathsf{eqtd}_k) \leftarrow \mathsf{SimCom}(\mathsf{ck}_k, \mathsf{td}_k)$ instead of committing to R_k. In $\mathcal{O}^{\mathsf{DEval}_2}$ for user $k = k_{\$}$, the game samples $R_k = R_{0,k} + R_{1,k}$ for $r_{b,k} \leftarrow \mathbb{Z}_p$ and $R_{b,k} = r_{b,k}G$.

Then, it equivocates the commitment cmt_k to obtain an opening $\mathsf{decmt}_k \leftarrow \mathsf{Equivocate}(\mathsf{td}_k, \mathsf{cmt}_k, \mathsf{eqtd}_k, R_k)$ to R_k for cmt_k. Otherwise, the game proceeds as before. We have under equivocability of COM that

$$|\varepsilon_{3.6.1} - \varepsilon_{3.7.1}| \leq \mathsf{negl}(\lambda).$$

Remark 7. (a) There is a one-to-one correspondence between cmt_i and R_i in $\mathcal{O}^{\mathsf{AggEval}}$ for all users $i \in \mathcal{S}$, *i.e.*, for each cmt_i there exists at most one R_i such that the adversary A can invoke $\mathcal{O}^{\mathsf{AggEval}}$ without abort. (b) Furthermore, the value $R_{k_\$}$ is sampled at random in $\mathcal{O}^{\mathsf{DEval}_2}$ *after* all other contributions R_j for $j \in \mathcal{S} \setminus \{k_\$\}$ are determined.

The one-to-one correspondence stated in (a) holds for dishonest users $i \in \mathcal{S}_\mathcal{C}$ due to the abort condition added in game 3.4, and for honest users $i \in \mathcal{S}_\mathcal{H}$ due to the fact that cmt_i and R_i are signed via σ_i in $\mathcal{O}^{\mathsf{DEval}_2}$ and that i never samples the same commitment twice (cf. game 3.3 and remark 5). Let us now argue why (b) holds. The correspondence between cmt_i and R_i for $i \neq k_\$$ is established in $\mathcal{O}^{\mathsf{DEval}_1}$ for honest users $i \in \mathcal{S}_\mathcal{H} \setminus \{k_\$\}$, and in the input to the call to $\mathcal{O}^{\mathsf{DEval}_2}$ for malicious users $i \in \mathcal{S}_\mathcal{C}$ because ck_i is setup in extractable mode. On the other hand, the $R_{k_\$}$ is sampled only during the execution of $\mathcal{O}^{\mathsf{DEval}_2}$.

Finally, let us upper bound the probability $\varepsilon_{3.7.1}$, *i.e.*, Case 1 occurs in game 3.7.1. If user k aggregated twice to Y, then there exist two sessions sid and sid' with commitments $(\mathsf{cmt}_j)_{j\in\mathcal{S}}, (\mathsf{cmt}'_j)_{j\in\mathcal{S}'}$, values x, x', user sets $\mathcal{S}, \mathcal{S}'$, and openings $(R_j)_{j\in\mathcal{S}}, (R'_j)_{j\in\mathcal{S}'}$, respectively, in which user k aggregated to Y. Due to the game hops above (see remark 7), the committed values R_i for cmt_i are determined for $i \neq k$ before user k samples its value R_k in $\mathcal{O}^{\mathsf{DEval}_2}$. Furthermore, the value x is determined as it is part of the state of user k in the corresponding $\mathcal{O}^{\mathsf{DEval}_2}$ call. Analogously, this holds for value R'_i in cmt'_i and x' for $i \neq k$ in the corresponding call to $\mathcal{O}^{\mathsf{DEval}_2}$.

Without loss of generality, let us assume that the call to $\mathcal{O}^{\mathsf{DEval}_2}$ for user k occurs first in session sid', then for session sid. In $\mathcal{O}^{\mathsf{DEval}_2}$ in session sid' after the game samples R'_k, the value $Y' = x'G + \sum_{j\in\mathcal{S}'} R'_j$ in $\mathcal{O}^{\mathsf{AggEval}}$ is information-theoretically determined (cf. remark 7). Further, in $\mathcal{O}^{\mathsf{DEval}_2}$ in session sid, the value R_k is sampled uniformly at random, whereas the other values R_j for $j \in \mathcal{S} \setminus \{k\}$ are already determined (again, by remark 7). This determines $Y = xG + \sum_{j\in\mathcal{S}} R_j$. Importantly, at this point Y' is already determined, therefore it holds that

$$\Pr[Y = Y'] = \Pr[R_k = Y' - xG - \sum_{j\in\mathcal{S}\setminus\{k\}} R_j] = \mathsf{negl}(\lambda).$$

Since we have polynomially many possible sessions (let CTR be an upper bound on the number of sessions) a union bound yields

$$\begin{aligned}\varepsilon_{3.7.1} &\leq \sum_{\mathsf{sid},\mathsf{sid}'\in[\mathsf{CTR}]} \Pr[Y = Y' \text{ for user } k_\$,\ \text{sessions } \mathsf{sid}, \mathsf{sid}'] \\ &\leq \mathsf{CTR}^2 \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda).\end{aligned}$$

To conclude the bound on the probability of Case 1 in Game 3.4:

$$\varepsilon_{3.4.1} \leq N \cdot \varepsilon_{3.5.1} \leq N \cdot (\varepsilon_{3.7.1} + \mathsf{negl}(\lambda)) = \mathsf{negl}(\lambda).$$

Case 2. Let us turn towards Case 2. If this case occurs, then there exist two sessions sid and sid' with commitments $(\mathsf{cmt}_j)_{j\in\mathcal{S}}, (\mathsf{cmt}'_j)_{j\in\mathcal{S}'}$, values x, x', user sets $\mathcal{S}, \mathcal{S}'$, and openings $(R_j)_{j\in\mathcal{S}}, (R'_j)_{j\in\mathcal{S}'}$, respectively, in which user $k \in \mathcal{S}_\mathcal{H} \subseteq \mathcal{S}$ aggregated to Y and user $k' \in \mathcal{S}'_\mathcal{H} \subseteq \mathcal{S}'$ aggregated to Y with $\mathcal{S} \neq \mathcal{S}'$. and $k \neq k'$.

Hence, if Case 2 is true, then by remark 6 the oracle $\mathcal{O}^{\mathsf{DEval}_2}$ must have been invoked for user k and user k' with inputs $(\mathsf{cmt}_j)_{j\in\mathcal{S}}$ and $(\mathsf{cmt}'_j)_{j\in\mathcal{S}'}$, respectively. Note that these calls determine R_k and $R'_{k'}$, respectively. We assume that the corresponding $\mathcal{O}^{\mathsf{DEval}_2}$ call for user k occurred *after* the one for user k'.[10]

As in Case 1, we guess user $k_\$ = k$ initially, where k is defined as above. We again define intermediate games 3.5.2 to 3.7.2 and corresponding probabilities $\varepsilon_{3.5.2}$ to $\varepsilon_{3.7.2}$ analogously to Case 1 to abort if the guess is incorrect and to sample $k_\$$'s contributions $R_{k_\$}$ in $\mathcal{O}^{\mathsf{DEval}_2}$ via equivocation. For conciseness, we only sketch the games below.

Game 3.5.2 The game samples $k_\$ \leftarrow \mathcal{H}$ initially and once Case 2 occurs for the first time (if it occurs at all), the game aborts if $k \neq k_\$$.

Game 3.6.2 The game sets up the commitment key $\mathsf{ck}_{k_\$}$ of $k_\$$ in equivocal mode.

Game 3.7.2 The game samples the nonce R_k of user $k = k_\$$ in $\mathcal{O}^{\mathsf{DEval}_2}$ instead of $\mathcal{O}^{\mathsf{DEval}_1}$.

We note that remark 7 holds in Game 3.7.2 analogously.

As in Case 1, we can show that

$$\varepsilon_{3.4.2} \leq N \cdot (\varepsilon_{3.7.2} + \mathsf{negl}(\lambda)).$$

To upper bound the probability of $\varepsilon_{3.7.2}$, let us distinguish two further subcases:

Case 2.1: $k \notin \mathcal{S}'$**.** In that case, the value Y' in $\mathcal{O}^{\mathsf{AggEval}}$ is determined in the $\mathcal{O}^{\mathsf{DEval}_2}$ call for user k' which opens cmt'_j to R'_k as all users $i \in \mathcal{S}$ in session sid' have extractable commitment keys (cf. remarks 6 and 7). Later, the $\mathcal{O}^{\mathsf{DEval}_2}$ call determines R_k, and therefore Y, which is sampled at random (cf. remark 7). We have

$$\Pr[Y = Y'] = \Pr[R_k = Y' - xG - \sum_{j\in\mathcal{S}\setminus\{k_\$\}} R_j] = \mathsf{negl}(\lambda).$$

Similar to Case 1, since there are polynomially many sessions and users, we obtain

$$\Pr[\text{Case 2.1 in Game 3.7.2}] = \mathsf{negl}(\lambda).$$

[10] The case that the corresponding $\mathcal{O}^{\mathsf{DEval}_2}$ call for user k occurred *before* the one for user k' can be argued analogously, but with inverse roles, i.e., we need to guess user k' instead of k and analyse the cases for $k' \notin \mathcal{S}$ and $k \in \mathcal{S}$.

Case 2.2: $k \in \mathcal{S}'$**.** Intuitively, it must hold that user k and user k' aggregate to the same value Y in session sid' due to symmetry (cf. remark 6). Therefore, we could upper bound the probability that $Y = Y'$ as in Case 2.1. However, such an explicit $\mathcal{O}^{\mathsf{AggEval}}$ call for user k in session sid' does not necessarily occur. Hence, we instead argue through the entropy of either R_k or R'_k, depending on which value is determined first through $\mathcal{O}^{\mathsf{DEval}_2}$.
Since user k' aggregated to Y', by remark 6, there exists a call to $\mathcal{O}^{\mathsf{DEval}_2}$ on input $(\mathsf{cmt}'_j)_{j\in\mathcal{S}'}$ for user k in which user k opens cmt'_k to R'_k. This call determines Y' by remarks 6 and 7.
Let us first argue that Y and Y' are determined in two different $\mathcal{O}^{\mathsf{DEval}_2}$ calls. Since $\mathcal{S}' \neq \mathcal{S}$, we have $\mathsf{cmt}_k \neq \mathsf{cmt}'_k$ due to remark 5). Both of those commitment must have been opened in two distinct $\mathcal{O}^{\mathsf{DEval}_2}$ calls.
The call that determines Y could happen either before or after the call that determines Y'. If the latter call occurs first, the value Y' is determined before R_k is uniformly sampled, and vice versa. Therefore, it follows as above

$$\Pr[Y = Y'] = \mathsf{negl}(\lambda).$$

Similar to Case 1, since there are polynomially many sessions and users, we obtain

$$\Pr[\text{Case 2.2 in Game 3.7.2}] = \mathsf{negl}(\lambda).$$

In conclusion, we have

$$\begin{aligned}\varepsilon_{3.4.2} &\leq N \cdot (\varepsilon_{3.7.2} + \mathsf{negl}(\lambda)) \\ &\leq N \cdot (\Pr[\text{Case 2.1 in Game 3.7.2}] + \Pr[\text{Case 2.2 in Game 3.7.2}] + \mathsf{negl}(\lambda)) \\ &= \mathsf{negl}(\lambda).\end{aligned}$$

Finally, combining Cases 1 and 2 now yields

$$\varepsilon_{3.4} \leq \varepsilon_{3.4.1} + \varepsilon_{3.4.2} = \mathsf{negl}(\lambda).$$

6.3 Equivocable and Extractable Commitment from DDH

We employ the dual-mode commitment scheme from [GS08] based on ElGamal encryption. It is shown to be extractable (for bit-messages) and perfectly-hiding (depending on which setup is employed). We give a brief sketch and show that it is also equivocable [11]. The extractable setup employs the commitment key

$$\mathsf{CK} = \begin{pmatrix} d & de \\ df & def \end{pmatrix} \cdot G,$$

where $d, e, f \leftarrow \mathbb{Z}_p$. To commit to a bit $b \in \{0, 1\}$, sample $\mathbf{r} \leftarrow \mathbb{Z}_p^2$ and set

$$\mathbf{C} := (0, bG) + \mathbf{r}^\intercal \mathsf{CK}.$$

[11] While we believe that this is folklore, we could not find a reference. We provide the argument for completeness.

The vector $\mathbf{r}$ serves as decommitment and verification passes if the above equation holds. To extract the bit b, output $b = 0$ if $0 = (-e, 1)\mathbf{C}^\intercal$, else output 1. For equivocation, set $\mathsf{CK} := \mathbf{X} \cdot G$ for $\mathbf{X} \leftarrow \mathbb{Z}_p^{2\times 2}$ such that $\mathsf{det}(\mathbf{X}) \neq 0$. Note that the setup is indistinguishability under DDH. The SimCom algorithm outputs a commitment $\mathbf{C}$ to 0, as well as the randomness $\mathsf{eqtd} = \mathbf{r}$ used to generate $\mathbf{C}$. To equivocate to 0, simply output $\mathbf{r}$. To open $\mathbf{C}$ to 1, output

$$\mathbf{r}^* := \mathbf{r} - \left(\mathbf{X}^{-1}\right)^\intercal \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Observe that $\mathbf{r}^*$ is well-distributed. Further, we have

$$\begin{aligned}(0, G) + (\mathbf{r}^*)^\intercal \,\mathsf{CK} &= (0, G) + \left(\mathbf{r} - \left(\mathbf{X}^{-1}\right)^\intercal \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right)^\intercal \mathbf{X} \cdot G \\ &= (0, G) + \mathbf{C} - \left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}^\intercal \cdot \mathbf{X}^{-1}\mathbf{X}\right) \cdot G \\ &= \mathbf{C}\end{aligned}$$

The commitment allows to commit to bits. Therefore, to commit to group elements $R \in \mathbb{G}$ (as required for our TCH construction), we require p commitments in total to commit to the binary representation of R. A straightforward optimization is to commit to the B-ary decomposition for $B = \mathsf{poly}(\lambda)$ of the bit-representation (when interpreted as binary number), similar to the encryption scheme in [KRS23, Section 6]. Then, the commitment to R is of size $2\lceil \log_B p \rceil$ group elements, however, extraction requires time $\mathcal{O}(\sqrt{B})$. As finding a more efficient instantiation is out-of-scope, we leave optimizations for future work.

Acknowledgments

We would like to thank the anonymous reviewers of TCC 2025 for their helpful feedback, in particular for spotting a mistake in a previous version of the TCH instantiation. C.B. is supported by the SNSF Ambizione grant No. PZ00P2_216140.

References

AMVA17. Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. In *2017 IEEE European Symposium on Security and Privacy*, pages 111–126. IEEE Computer Society Press, April 2017.

BCK+22. Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Cham, August 2022.

BGHJ24. Manuel Barbosa, Kai Gellert, Julia Hesse, and Stanislaw Jarecki. Bare PAKE: Universally composable key exchange from just passwords. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 183–217. Springer, Cham, August 2024.

BL22. Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 193–207. ACM Press, November 2022.

BLT+24. Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from DDH with full adaptive security. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 429–459. Springer, Cham, May 2024.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003.

BP23. Luis Brandao and Rene Peralta. NIST first call for multi-party threshold schemes, 2023.

BR08. Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to VSH. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 125–142. Springer, Berlin, Heidelberg, December 2008.

BSW06. Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 229–240. Springer, Berlin, Heidelberg, April 2006.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CDK+17. Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 152–182. Springer, Berlin, Heidelberg, March 2017.

Che25. Yanbo Chen. Dazzle: Improved adaptive threshold signatures from DDH. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part III*, volume 15676 of *LNCS*, pages 233–261. Springer, Cham, May 2025.

CKGW23. Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A Wood. Two-round threshold schnorr signatures with frost. Technical report, Internet-Draft draft-irtf-cfrg-frost-10, Internet Engineering Task Force , 2023.

CKM21. Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021.

CKM23. Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Cham, August 2023.

Des88. Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Berlin, Heidelberg, August 1988.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990.

DKM+24. Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024.

DR24. Sourav Das and Ling Ren. Adaptively secure BLS threshold signatures from DDH and co-CDH. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 251–284. Springer, Cham, August 2024.

DSSS19. David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *NDSS 2019*. The Internet Society, February 2019.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Berlin, Heidelberg, May 1996.

GJKR07. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Berlin, Heidelberg, April 2008.

KG20. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.

KR00. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS 2000*. The Internet Society, February 2000.

KRS23. Shuichi Katsumata, Michael Reichle, and Yusuke Sakai. Practical round-optimal blind signatures in the ROM from standard assumptions. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 383–417. Springer, Singapore, December 2023.

KRT24. Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024.

LJY14. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014.

LJY16. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theor. Comput. Sci.*, 645:1–24, 2016.

LNÖ25. Anja Lehmann, Phillip Nazarian, and Cavit Özbay. Stronger security forăthreshold blind signatures. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 335–364, Cham, 2025. Springer Nature Switzerland.

MMS+24. Aikaterini Mitrokotsa, Sayantan Mukherjee, Mahdi Sedaghat, Daniel Slamanig, and Jenit Tomy. Threshold structure-preserving signatures: Strong and adaptive security under standard assumptions. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part I*, volume 14601 of *LNCS*, pages 163–195. Springer, Cham, April 2024.

NT24. Sela Navot and Stefano Tessaro. One-more unforgeability for multi - and threshold signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 429–462. Springer, Singapore, December 2024.

RRJ+22. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous Schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Sho00. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Berlin, Heidelberg, May 2000.

SPW07. Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In Masayuki Abe, editor, *CT-RSA 2007*, volume 4377 of *LNCS*, pages 357–371. Springer, Berlin, Heidelberg, February 2007.

TZ23. Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, April 2023.

Supplementary Material

A Additional Preliminaries

We recall some standard notions.

A.1 Signatures

In the following, we recall the definition of single-user and threshold signatures.

Definition 8 (Signatures). *A signature scheme Σ consists of four algorithms* Setup, KeyGen, Sign, Vrfy *that have the following syntax:*

- $\mathsf{Setup}(1^\lambda)$*: Takes as input a security parameter (in unary) and outputs public parameters* pp*, which include λ, as well as message and signature spaces $\mathcal{M}$ and $\mathcal{S}$.*
- $\mathsf{KeyGen}(\mathsf{pp})$*: Takes as input public parameters* pp *and outputs a public key* pk *and a secret key* sk*. The public key* pk *includes the public parameters* pp *and the secret key* sk *includes* pk*.*
- $\mathsf{Sign}(m, \mathsf{sk})$*: Takes as input a message $m \in \mathcal{M}$ and a secret key* sk *and outputs a signature σ.*
- $\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$*: Takes as input a public key* pk*, a message $m \in \mathcal{M}$ and a signature σ, and outputs a bit b.*

Definition 9 (Correctness). *A signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ satisfies* correctness *if for all $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, $m \in \mathcal{M}$ it holds*

$$\Pr[\mathsf{Vrfy}(\mathsf{pk}, m, \sigma) = 0 \mid \sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk})] \leq \mathsf{negl}(\lambda).$$

Definition 10 (Unforgeability for signatures). *A signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ satisfies unforgeability if for any* PPT *adversary* A*, we have*

$$\mathsf{Adv}^{\mathsf{uf}}_{\Sigma,\mathsf{A}}(1^\lambda) := \Pr[\mathsf{Game}^{\mathsf{uf}}_{\Sigma,\mathsf{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the game $\mathsf{Game}^{\mathsf{uf}}_{\mathsf{TS},\mathsf{A}}$ is defined as below.

$\underline{\mathsf{Game}^{\mathsf{uf}}_{\Sigma,\mathsf{A}}(1^\lambda)}$

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
3. $\mathsf{M} := \emptyset$ // *initialize set of queried messages*
4. $(m^*, \sigma^*) \leftarrow \mathsf{A}^{\mathcal{O}^{\mathsf{Sign}}}(\mathsf{pk})$
5. **output** $[\![m^* \notin \mathsf{M} \ \wedge \ \mathsf{Vrfy}(\mathsf{pk}, m^*, \sigma^*) = 1]\!]$

$\underline{\mathcal{O}^{\mathsf{Sign}}(m)}$ // $m \in \mathcal{M}$

1. $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk})$
2. $\mathsf{M} := \mathsf{M} \cup \{m\}$
3. **output** σ

A.2 Collision resistant hash functions

We recall the definition of collision resistant hash functions.

Definition 11. *A hash function* h *with output length* ℓ *consists of three efficient algorithms* Setup, Gen *and* Eval *with the following syntax:*

- $\mathsf{Setup}(1^\lambda)$*: Takes as input a security parameter (in unary), and outputs public parameters* pp_h*,*
- $\mathsf{Gen}(\mathsf{pp}_\mathsf{h})$*: Given public parameters, outputs a description* f_h *of a hash function* $\{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$*. We assume* f_h *contains* pp_h*.*
- $\mathsf{Eval}(f_\mathsf{h}, x)$*: Given a description of a hash function* f_h *and an input string* $x \in \{0,1\}^*$*, outputs some* $y \in \{0,1\}^{\ell(\lambda)}$*. We often write* $f_\mathsf{h}(x)$ *to denote* $\mathsf{Eval}(f_\mathsf{h}, x)$*.*

Definition 12 (Collision resistance). *We say that a hash function* h *is collision resistant, if for any* PPT *adversary* A *and* $\mathsf{pp}_\mathsf{h} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $f_\mathsf{h} \leftarrow \mathsf{Gen}(\mathsf{pp}_\mathsf{h})$ *we have*

$$\Pr[x \neq x' \wedge f_\mathsf{h}(x) = f_\mathsf{h}(x') \mid (x, x') \leftarrow \mathsf{A}(f_\mathsf{h})] \leq \mathsf{negl}(\lambda)$$

A.3 Commitments

Our scheme will rely on extractable and equivocable commitments; we recall their definition in the following.

Definition 13 (Commitment). *A commitment* COM *consists of three efficient algorithms* $\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}$ *that have the following syntax:*

- $\mathsf{Setup}(1^\lambda)$*: Takes as input a security parameter (in unary), and outputs a commitment key* ck*. We assume that* ck *specifies a message space* $\mathcal{M}$*.*
- $\mathsf{Commit}(\mathsf{ck}, m)$*: Given commitment key* ck *and a message* m*, outputs a commitment* cmt *and decommitment* decmt*.*
- $\mathsf{Verify}(\mathsf{ck}, \mathsf{cmt}, m, \mathsf{decmt})$*: Given commitment key* ck*, commitment* cmt*, message* m *and decommitment* decmt*, outputs a bit* b*.*

Definition 14 (Correctness). *A commitment scheme* $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ *satisfies* correctness *if for all* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $m \in \mathcal{M}$ *it holds*

$$\Pr[\mathsf{Verify}(\mathsf{ck}, \mathsf{cmt}, m, \mathsf{decmt}) = 0 \mid (\mathsf{cmt}, \mathsf{decmt}) \leftarrow \mathsf{Commit}(\mathsf{ck}, m)] \leq \mathsf{negl}(\lambda).$$

We define the extraction and equivocation properties. Note that these properties imply the standard notions binding and hiding, respectively.

Definition 15 (Extractability). *A commitment scheme* $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ *is* extractable *if there exists a* PPT *algorithm* $\mathsf{ExtSetup}$ *and a deterministic* PT *algorithm* $\mathsf{Extract}$ *such that for any* PPT *adversary* A *it holds that*

$$\Pr\left[\mathsf{A}(\mathsf{ck}_b) = b \,\middle|\, \begin{array}{c} b \leftarrow \{0,1\}, \\ (\mathsf{ck}_0, \mathsf{td}) \leftarrow \mathsf{ExtSetup}(1^\lambda), \\ \mathsf{ck}_1 \leftarrow \mathsf{Setup}(1^\lambda) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

and

$$\Pr\left[\begin{array}{c} m \neq m' \wedge \\ \mathsf{Verify}(\mathsf{ck}, \mathsf{cmt}, m, \mathsf{decmt}) = 1 \end{array} \,\middle|\, \begin{array}{c} (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ExtSetup}(1^\lambda), \\ (\mathsf{cmt}, m, \mathsf{decmt}) \leftarrow \mathsf{A}(\mathsf{ck}) \\ m' \leftarrow \mathsf{Extract}(\mathsf{td}, \mathsf{cmt}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

Definition 16 (Equivocability). *A commitment scheme* $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ *is* equivocable *if there exists* PPT *algorithms* $\mathsf{EqvSetup}, \mathsf{SimCom}$ *and* $\mathsf{Equivocate}$ *such that for any stateful* PPT *adversary* A *it holds that*

$$\Pr\left[\mathsf{A}(\mathsf{ck}_b) = b \,\middle|\, \begin{array}{c} b \leftarrow \{0,1\}, \\ (\mathsf{ck}_0, \mathsf{td}) \leftarrow \mathsf{EqvSetup}(1^\lambda), \\ \mathsf{ck}_1 \leftarrow \mathsf{Setup}(1^\lambda) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

and

$$\Pr[\mathsf{Game}^{\mathsf{com-eq}}_{\mathsf{COM},\mathsf{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the game $\mathsf{Game}^{\mathsf{com-eq}}_{\mathsf{COM}}$ *is defined as:*

$\underline{\mathsf{Game}^{\mathsf{com-eq}}_{\mathsf{COM},\mathsf{A}}(1^\lambda)}$

1. $b \leftarrow \{0,1\}$
2. $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{EqvSetup}(1^\lambda)$
3. $b' \leftarrow \mathsf{A}^{\mathcal{O}_b^{\mathsf{Commit}}}(\mathsf{ck})$
4. **output** $[\![b' = b]\!]$

$\underline{\mathcal{O}_0^{\mathsf{Commit}}(m)}$ // $m \in \mathcal{M}$

1. $(\mathsf{cmt}, \mathsf{decmt}) \leftarrow \mathsf{Commit}(\mathsf{ck}, m)$
2. **output** $(\mathsf{cmt}, \mathsf{decmt})$

$\underline{\mathcal{O}_1^{\mathsf{Commit}}(m)}$

1. $(\mathsf{cmt}, \mathsf{eqtd}) \leftarrow \mathsf{SimCom}(\mathsf{ck}, \mathsf{td})$
2. $\mathsf{decmt} \leftarrow \mathsf{Equivocate}(\mathsf{td}, \mathsf{cmt}, \mathsf{eqtd}, m)$
3. **output** $(\mathsf{cmt}, \mathsf{decmt})$

A.4 Pseudorandom functions

We recall the definition of pseudorandom functions. Note that we define a multi-challenge security game.

Definition 17. *Let $n_\lambda, m_\lambda, \ell_\lambda > 0$ be integers. We say that a deterministic polynomial time algorithm $\mathsf{PRF} : \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}^m$ is a pseudorandom function if for any PPT adversary A we have*

$$\Pr[\mathsf{Game}^{\mathsf{prf}}_{\mathsf{PRF},\mathsf{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathsf{Game}^{\mathsf{prf}}$ is defined as:

$\underline{\mathsf{Game}^{\mathsf{prf}}_{\mathsf{PRF},\mathsf{A}}(1^\lambda)}$

1. $\mathsf{T}[\cdot] := \bot$.
2. $K \leftarrow \{0,1\}^\ell$
3. $b \leftarrow \{0,1\}$
4. $b' \leftarrow \mathsf{A}^{\mathcal{O}_{\mathsf{PRF}}}(1^\lambda)$
5. **output** $[\![b' = b]\!]$

$\underline{\mathcal{O}_{\mathsf{PRF}}(x)}$ // $x \in \{0,1\}^n$

1. $y_0 = \mathsf{PRF}_K(x)$
2. *if* $\mathsf{T}[x] = \bot$, *then* $\mathsf{T}[x] \leftarrow \{0,1\}^m$.
3. $y_1 = \mathsf{T}[x]$
4. **output** y_b

A.5 Chameleon hash functions

Chameleon hash functions were introduced by Krawczyk and Rabin [KR00].

Definition 18 (Chameleon hash function). *A chameleon hash function* CH *consists of four algorithms* Setup, Gen, Eval, $\mathsf{TrapColl}$ *that have the following syntax*

- $\mathsf{Setup}(1^\lambda)$*: Takes as input a security parameter (in unary) and outputs public parameters* pp *which includes* λ.
- $\mathsf{Gen}(\mathsf{pp})$*: Takes as input public parameters* pp *and outputs the description* ch *of a function* $\mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ *(that includes* pp*) as well as a trapdoor* τ *for* ch.
- $\mathsf{Eval}(\mathsf{ch}, x, r)$*: Takes as input the description of a chameleon hash function* ch*, an input* $x \in \mathcal{X}$ *and random coins* $r \in \mathcal{R}$*, and outputs some* $y \in \mathcal{Y}$.
- $\mathsf{TrapColl}(\mathsf{ch}, \tau, x, r, x^*)$*: Takes as in put the description of* ch*, two inputs* $x, x^* \in \mathcal{X}$ *and random coins* $r \in \mathcal{R}$*, and outputs* $r^* \in \mathcal{R}$.

Definition 19 (Correctness). *A chameleon hash function* $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{TrapColl})$ *satisfies* correctness *if for all* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{ch}, \tau) \leftarrow \mathsf{Gen}(\mathsf{pp})$, *all* $x, x^* \in \mathcal{X}$ *and* $r \in \mathcal{R}$

$$\Pr[\mathsf{Eval}(\mathsf{ch}, x^*, r^*) \neq \mathsf{Eval}(\mathsf{ch}, x, r) \mid r^* \leftarrow \mathsf{TrapColl}(\mathsf{ch}, \tau, x, r, x^*)] \leq \mathsf{negl}(\lambda).$$

We require a stronger notion of collision resistance than the original definition in [KR00]: while in our definition any pair $(x, r) \neq (x^*, r^*)$ with $\mathsf{Eval}(\mathsf{ch}, x, r) = \mathsf{Eval}(\mathsf{ch}, x^*, r^*)$ counts as a collision, in the definition of [KR00], $x \neq x^*$ is required.

Definition 20 (Collision resistance). *A chameleon hash function* $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{TrapColl})$ *is* collision-resistant *if for any* PPT *adversary* A

$$\Pr\left[\begin{array}{c|c} \mathsf{Eval}(\mathsf{ch}, x, r) = \mathsf{Eval}(\mathsf{ch}, x^*, r^*) & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \wedge & (\mathsf{ch}, \tau) \leftarrow \mathsf{Gen}(\mathsf{pp}), \\ (x, r) \neq (x^*, r^*) & (x, r, x^*, r^*) \leftarrow \mathsf{A}(\mathsf{ch}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 21 (Indistinguishability). *A chameleon hash function* $\mathsf{CH} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval}, \mathsf{TrapColl})$ *with input space* $\mathcal{X}$ *and randomness space* $\mathcal{R}$ *satisfies* indistinguishability *if for any* PPT *adversary* A

$$\Pr[\mathsf{Game}^{\mathsf{ch-ind}}_{\mathsf{CH},\mathsf{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the game $\mathsf{Game}^{\mathsf{ch-ind}}_{\mathsf{CH}}$ *is defined as:*

$\underline{\mathsf{Game}^{\mathsf{ch-ind}}_{\mathsf{CH},\mathsf{A}}(1^\lambda)}$

1. $b \leftarrow \{0,1\}$
2. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
3. $(\mathsf{ch}, \tau) \leftarrow \mathsf{Gen}(\mathsf{pp})$
4. $b' \leftarrow \mathsf{A}^{\mathcal{O}_b^{\mathsf{TrapColl}}}(\mathsf{pp}, \mathsf{ch})$
5. **output** $[\![b' = b]\!]$

$\underline{\mathcal{O}_0^{\mathsf{TrapColl}}(x^*)}$ // $x^* \in \mathcal{X}$

1. $r \leftarrow \mathcal{R}$
2. **output** r

$\underline{\mathcal{O}_1^{\mathsf{TrapColl}}(x^*)}$ // $x^* \in \mathcal{X}$

1. $r \leftarrow \mathcal{R}$, $x \leftarrow \mathcal{X}$
2. $r^* \leftarrow \mathsf{TrapColl}(\mathsf{ch}, \tau, x, r, x^*)$
3. **output** r^*

B Generic Transformation — Full Details

In this section, we describe the $\Sigma.\mathsf{Sign}$ protocol in full details. For the definition of $\Sigma.\mathsf{Setup}$, $\Sigma.\mathsf{Gen}$ and $\Sigma.\mathsf{Vrfy}$, as well as the high-level description of the whole construction, we refer the reader to section 5.

The signing protocol $\Sigma.\mathsf{Sig}$ consists of multiple algorithms $(\{\Sigma.\mathsf{Sign}_r\}_{r \in R}$, $\Sigma.\mathsf{SigAgg})$, where $\Sigma.\mathsf{Sign}_r$ describes the local computation of an honest user in round r (we refer to these as *round algorithms*), and $\Sigma.\mathsf{SigAgg}$ defines how to aggregate the protocol transcript into a final signature.

Each round algorithm takes as input round messages of all parties from the previous round $\{\mathsf{rm}_s\}_{s \in \mathcal{S}}$, and the state st of the party executing the algorithm. The first round algorithm is executed on empty round messages and the initial state of each party consists of the inputs into the protocol; concretely, the index of the party k executing the algorithms, the set of participating users $\mathcal{S}$, the message being signed m and the context ctx_Σ in which the protocol is run. By executing a round algorithm $\Sigma.\mathsf{Sign}_r$ on a given set of round message from round $r-1$ and state of user k, the r-th round message of user k is defined and the state of the user k gets updated (by appending some values to the state including the round message of the user).

We use the following conventions to update a user's state. When we write "Add x to st", we formally mean that a new attribute of st is defined (under the name x) and the value of $\mathsf{st}.x$ is initially set to the value of x. When we write "Update x in st, we implicitly assume that $\mathsf{st}.x$ is already defined, but the value is now overwritten by the value of x. Despite the informality of the notation, we believe this does not cause confusion and eases the readability of the pseudo-code.

Each round algorithm begins by parsing a user's state (i.e. extracting the input values of the protocol), checking that the input round message of user k is consistent with what user k actually output in the previous round, and storing previous round messages of other parties into the state (this allows to extract a protocol transcript from a user's state). To avoid repeating these instructions in every round algorithm, we define a helper function OK which does exactly this, see left part of fig. 4.

Another helper function that we find useful to define separately is $\mathsf{Transcript}$. As the name suggests, it helps to parse a protocol transcript. It takes as input a keyword $\mathtt{str} \in \{\mathtt{DEval}, \mathtt{Sign}, \mathtt{TrapCol}\}$ and a user's state. Depending on the keyword, the function extracts relevant round messages from the user's state and outputs a transcript of the $\mathsf{H.DEval}$, $\tilde{\Sigma}.\mathsf{Sig}$ or $\mathsf{H.TrapColl}$ protocol respectively.

We are now prepared to present the full pseudo-code of the protocol in fig. 5. Let us stress that if st is set to $\perp$, the algorithm immediately aborts (outputs $\perp$ as round message and $\perp$ as the user's state). Moreover, we assume that if an algorithm gets $\perp$ as input (i.e., the input is undefined), it immediately outputs $\perp$ again.

$\underline{\mathsf{OK}(r, (\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})}$:
// Checks consistency and outputs main values stored in st

1. Extract $k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{ctx}_\Sigma$ from st
2. If $r = 1$:
   (a) $\exists s \in \mathcal{S}$ s.t. $\mathsf{rm}_s \neq \varepsilon$, then $\mathsf{st} \leftarrow \perp$ // Abort if some initial message non-empty
   Else
   (a) Extract $(k, r-1, \mathsf{rm})$ from st // get message sent by k in previous round
   (b) If $\mathsf{rm} \neq \mathsf{rm}_k$, then $\mathsf{st} \leftarrow \perp$ // Abort if k's message inconsistent
   (c) Else add $(s, r-1, \mathsf{rm}_s)$ to st for all $s \in \mathcal{S}$ // Store message from the previous round
3. Output $(k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{st}, \mathsf{ctx}_\Sigma)$

$\underline{\mathsf{Transcript}(\mathtt{str}, \mathsf{st})}$:
// Returns a transcript for an indicated distributed protocol

1. Extract $\mathcal{S}$ from st
2. Set r_0, r_1 depending on $\mathtt{str}$:
   - $\mathtt{DEval}$: $r_0 \leftarrow 1,\ r_1 \leftarrow R_e$
   - $\mathtt{Sign}$: $r_0 \leftarrow R_e + 1,\ r_1 \leftarrow R_e + R_S$
   - $\mathtt{TrapCol}$: $r_0 \leftarrow R_e + R_S + 1,\ r_1 \leftarrow R_e + R_S + R_t$
3. Extract $\{(s, j, \mathsf{rm}_{s,j})\}_{s\in\mathcal{S}, j\in[r_0,r_1]}$ from st
4. $\mathsf{trsc} \leftarrow (\mathsf{rm}_{s,j})_{s\in\mathcal{S}, j\in[r_0,r_1]}$
5. Output trsc

Fig. 4. Helper functions

$\Sigma.\mathsf{Sign}_i((\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})$, for $i \in \{1, \ldots, R_e\}$:
// i-th H.DEval round

1. $(k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{st}, \mathsf{ctx}_\Sigma) \leftarrow \mathsf{OK}(r, (\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})$
2. Extract $\mathsf{ch}_\mathsf{H}, x'$ from sk_k
3. If $i = 1$
   (a) Set $\mathsf{est} := (k, \mathcal{S}, \mathsf{ch}_\mathsf{H}, x')$
   (b) Add est to st
   Else extract est from st
4. Set $\mathsf{erm}_s := \mathsf{rm}_s$ for all $s \in \mathcal{S}$
5. $(\mathsf{est}, \mathsf{erm}) \leftarrow \mathsf{H.DEval}_i((\mathsf{erm}_s)_{s\in\mathcal{S}}, \mathsf{est})$
6. $\mathsf{rm} \leftarrow \mathsf{erm}$
7. Add (k, r, erm) to st and update est in st
8. Output $(\mathsf{st}, \mathsf{rm})$

$\Sigma.\mathsf{Sign}_{r=R_e+i}((\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})$, for $i \in \{2, \ldots, R_S\}$:
// i-th round of $\tilde{\Sigma}$.Sign

1. $(k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{st}, \mathsf{ctx}_\Sigma) \leftarrow \mathsf{OK}(r, (\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})$
2. Extract $\mathsf{ch}_\mathsf{H}, x'$ and $\tilde{\mathsf{sk}}_k$ from sk_k
3. If $i = 1$:
   (a) $\mathsf{trsc} \leftarrow \mathsf{Transcript}(\mathtt{DEval}, \mathsf{st})$
   (b) $y \leftarrow \mathsf{H.AggEval}(\mathsf{ch}_\mathsf{H}, \mathcal{S}, x', \mathsf{trsc})$
   (c) $\mathsf{sst} := (k, \tilde{\mathsf{sk}}_k, \mathcal{S}, y||m, \mathsf{ctx}_\Sigma)$
   (d) $\mathsf{srm}_s := \varepsilon$ for all $s \in \mathcal{S}$
   (e) Add y and sst to st
   Else
   (a) Extract sst from st
   (b) $\mathsf{srm}_s := \mathsf{rm}_s$ for all $s \in \mathcal{S}$
4. $(\mathsf{sst}, \mathsf{srm}) \leftarrow \tilde{\Sigma}.\mathsf{Sign}_i((\mathsf{srm}_s)_{s\in\mathcal{S}}, \mathsf{sst})$
5. $\mathsf{rm} \leftarrow \mathsf{srm}$
6. Add (k, r, srm) to st and update sst in st
7. Output $(\mathsf{st}, \mathsf{rm})$

$\Sigma.\mathsf{Sign}_{r=R_e+R_S+i}((\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})$, $i \in \{1, \ldots, R_t\}$:
// i-th H.DTrapColl round

1. $(k, \mathcal{S}, m, \mathsf{sk}_k, \mathsf{st}, \mathsf{ctx}_\Sigma) \leftarrow \mathsf{OK}(r, (\mathsf{rm}_s)_{s\in\mathcal{S}}, \mathsf{st})$
2. Extract $\tilde{\mathsf{pk}}, \tau_k, f_\mathsf{h}$ from sk_k
3. If $i = 1$:
   (a) Extract est, y from st
   (b) $\mathsf{trsc} \leftarrow \mathsf{Transcript}(\mathtt{Sign}, \mathsf{st})$
   (c) $\tilde{\sigma} \leftarrow \tilde{\Sigma}.\mathsf{SigAgg}(\mathsf{pk}, \mathcal{S}, y||m, \mathsf{trsc})$
   (d) $x := f_\mathsf{h}(\tilde{\sigma})$
   (e) $\mathsf{ctx}_\mathsf{t} := (\mathsf{ctx}_\Sigma, m)$
   (f) $\mathsf{tst} := (\mathsf{est}, \tau_k, x, \mathsf{ctx}_\mathsf{t})$
   (g) For $s \in \mathcal{S}$:
      i. Extract (s, R_e, erm)
      ii. $\mathsf{trm}_s := \mathsf{erm}$
   (h) Add $\tilde{\sigma}, \mathsf{tst}$ to st
   Else
   (a) Extract tst from st
   (b) $\mathsf{trm_s} := \mathsf{rm_s}$ for all $s \in \mathcal{S}$
4. $(\mathsf{tst}, \mathsf{trm}) \leftarrow \mathsf{H.DTrapColl}_\mathsf{i}((\mathsf{trm_s})_{s\in\mathcal{S}}, \mathsf{tst})$
5. $\mathsf{rm} \leftarrow \mathsf{trm}$
6. Add (k, r, trm) to st and update tst in st
7. If $r = R_e + R_S + R_t$:
   – $\mathsf{ssid}_\mathsf{t} := \mathsf{tst}$
   – Set $\mathsf{st} := (\mathsf{ssid}_\mathsf{t}, y)$
8. Output $(\mathsf{st}, \mathsf{rm})$

$\Sigma.\mathsf{SigAgg}(\mathsf{pk}, \mathcal{S}, m, (\mathsf{rm}_{s,r})_{s\in\mathcal{S}, r\in[R]})$:
// Signature aggregation algorithm

1. $\mathsf{st} \leftarrow \varepsilon$
2. For all $s \in \mathcal{S}, r \in [R]$:
   add $(s, r, \mathsf{rm}_{s,r})$ to st
3. Extract $\tilde{\mathsf{pk}}, \mathsf{ch}_\mathsf{H}, x'$ from pk
4. $\mathsf{trsc}^E \leftarrow \mathsf{Transcript}(\mathtt{DEval}, \mathsf{st})$
5. $\mathsf{trsc}^S \leftarrow \mathsf{Transcript}(\mathtt{Sign}, \mathsf{st})$
6. $\mathsf{trsc}^C \leftarrow \mathsf{Transcript}(\mathtt{TrapCol}, \mathsf{st})$
7. $y \leftarrow \mathsf{H.AggEval}(\mathsf{ch}_\mathsf{H}, \mathcal{S}, x', \mathsf{trsc}^E)$
8. $\tilde{\sigma} \leftarrow \tilde{\Sigma}.\mathsf{SigAgg}(\tilde{\mathsf{pk}}, \mathcal{S}, y||m, \mathsf{trsc}^S)$
9. $r_\mathsf{H} \leftarrow \mathsf{H.AggColl}(\mathsf{ch}_\mathsf{H}, \mathcal{S}, \mathsf{trsc}^\mathsf{H})$
10. Output $\sigma = (\tilde{\sigma}, r_\mathsf{H})$

Fig. 5. Round-by-round signature protocol. Total number of rounds: $R = R_e + R_S + R_t$.